Detection of automatically generated texts

Minh Tien Nguyen

Communauté UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTÉ UNIVERSITÉ GRENOBLE ALPES

Spécialité : Mathématiques et Informatique

Arrêté ministériel : 25 mai 2016

Présentée par

Minh Tien NGUYEN

Thèse dirigée par **Cyril LABBE**, Maitre de Conférence, UGA

préparée au sein du **Laboratoire d'Informatique de Grenoble** dans **l'École Doctorale Mathématiques, Sciences et technologies de l'information, Informatique**

Détection de textes générés automatiquement

Detection of Automatically Generated Texts

Thèse soutenue publiquement le **3 avril 2018**,
devant le jury composé de :

Monsieur CYRIL LABBE
MAITRE DE CONFERENCES, UNIVERSITE GRENOBLE ALPES, Directeur de thèse
Monsieur JACQUES SAVOY
PROFESSEUR, UNIVERSITE DE NEUCHATEL - SUISSE, Rapporteur
Monsieur GUILLAUME CABANAC
MAITRE DE CONFERENCES, UNIVERSITE TOULOUSE-III-PAUL-SABATIER, Rapporteur
Madame SYLVIE CALABRETTO
PROFESSEUR, INSA LYON, Président
Madame CATHERINE BERRUT
PROFESSEUR, UNIVERSITE GRENOBLE ALPES, Examinateur

Dedication

For my mother, my father, my friends.
Thanks you for believing and always being there for me.

Abstract

Automatically generated text has been used in numerous occasions with distinct intentions. It can simply go from generated comments in an online discussion to a much more mischievous task, such as manipulating bibliography information. So, this thesis first introduces different methods of generating free texts that resemble a certain topic and how those texts can be used. Therefore, we try to tackle multiple research questions. The first question is how and what is the best method to detect a fully generated document?

Then, we take it one step further to address the possibility of detecting a couple of sentences or a small paragraph of automatically generated text by proposing a new method to calculate sentences similarity using their grammatical structure. The last question is how to detect an automatically generated document without any samples. This is used to address the case of a new generator or a generator from which it is impossible to collect samples.

This thesis also deals with the industrial aspect of development. A simple overview of a publishing workflow from a high-profile publisher is presented. From there, an analysis is carried out to be able to best incorporate our method of detection into the production workflow.

In conclusion, this thesis has shed light on multiple important research questions about the possibility of detecting automatically generated texts in different settings. Besides the research aspect, important engineering work in a real-life industrial environment is also carried out to demonstrate that it is important to have real application along with fundamental research.

Résumé

Le texte généré automatiquement a été utilisé dans de nombreuses occasions à des buts différents. Il peut simplement passer des commentaires générés dans une discussion en ligne à une tâche beaucoup plus malveillante, comme manipuler des informations bibliographiques. Ainsi, cette thèse introduit d'abord différentes méthodes pour générer des textes libres ayant trait à un certain sujet et comment ces textes peuvent être utilisés. Par conséquent, nous essayons d'aborder plusieurs questions de recherche. La première question est comment et quelle est la meilleure méthode pour détecter un document entiérement généré.

Ensuite, nous irons un peu plus loin et montrer la possibilité de détecter quelques phrases ou un petit paragraphe de texte généré automatiquement en proposant une nouvelle méthode pour calculer la similarité des phrases en utilisant leur structure grammaticale. La dernière question est comment détecter un document généré automatiquement sans aucun échantillon, ceci est utilisé pour illustrer le cas d'un nouveau générateur ou d'un générateur dont il est impossible de collecter des échantillons dessus.

Cette thèse étudie également l'aspect industriel du développement. Un aperçu simple d'un flux de travail de publication d'un éditeur de premier plan est présenté. À partir de là, une analyse est effectuée afin de pouvoir intégrer au mieux notre méthode de détection dans le flux de production.

En conclusion, cette thèse a fait la lumière sur de multiples questions de recherche importantes concernant la possibilité de détecter des textes générés automatiquement dans différents contextes. En plus de l'aspect de la recherche, des travaux d'ingénierie importants dans un environnement industriel réel sont également réalisés pour démontrer qu'il est important d'avoir une application réelle pour accompagner une recherche fondamentale.

Acknowledgment

This research project was funded and accomplished in collaboration with Spriger - Nature. We are in deep gratitude to our associates at Springer - Nature, especially to everyone in the Process and Content Management department for their continuous support as well as constructive criticisms along the way.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the years, some questionable events have surfaced. It might have all begun with the so-called "Sokal affair" or "Sokal hoax." In 1996, Alan Sokal[1], a Physics professor at New York University and University College London, wrote a paper named "Transgressing the Boundaries: Towards a Transformative Hermeneutics of Quantum Gravity" [Sokal, 1996] and submitted it to *Social Text* - an academic journal of postmodern cultural studies.

In his paper, Sokal suggested that there is a progressive political implication to quantum gravity which is a field of theoretical physics with quantum mechanics. Furthermore, he also declared that "physical reality is fundamentally a social and linguistic construct" along with other ideological concepts. These claims are clearly nonsensical; however, they sound scientific and conform with the ideology of the journal.

Despite all the flaws, his paper was accepted and published in the "Science Wars" issue by Social Text without any peer review. At the same time as this publication, Sokal also revealed in another journal called Lingua Franca that the paper "Transgressing the Boundaries" was no more than a hoax. He said it was published without any review or consultation from any knowledgeable people in the field, while the editors at Social Text stand on their decision to publish it. This might raise some questions on what should be considered as appropriate to be published and how the review process should be implemented.

On the same note as Sokal, [Bohannon, 2013] created a set of nonsense papers in biology to re-verify the state of review in different open access journals. In these papers, he claimed that "Molecule X from lichen species Y inhibits the growth of cancer cell Z" where X, Y and Z were chosen at random from a pre-created database to create several unique, but quite similar papers. Furthermore, these papers contained numerous obvious flaws or even completely opposite explanations between a graph and its explanation. They are further modified by being translated into French and back to English to create an illusion that they were written by a non-native speaker.

However, when these papers were submitted to 304 different open access journals in the span of 8 months, surprisingly, despite the obvious flaws, 157 of these journals accepted the paper to be published. For the journals that accepted the paper, 106 performed some type of review but mostly focused exclusively on the paper's layout, formatting or language, all without any scientific concerns.

[1] `www.physics.nyu.edu/faculty/sokal/`

The results from this experiment show that for some of the "open access journals", the presentation of the paper seems to be the only important characteristic, and not the content nor the scientific value of the research. Or in other words, mass automatically generated papers where the layout is correct, but which contain little to no coherence ideal, would also be perfectly suited for these types of "journals."

Another attempt to manipulate bibliography was carried out by Ike Antkare [Labbé, 2010]. He was able to become one of the most highly cited authors on Google Scholar even though all his "research" was automatically generated. The author exploited Google Scholar's h-index algorithm using "research papers" which were automatically generated using SCIgen - An Automatic CS Paper Generator[2].

SCIgen is a computer program that can be used to create completely nonsensical scientific papers, but with perfect structure along with nice-looking tables, and graphs (Figure 1.1). SCIgen is capable of producing en masse a variety of pseudo computer science papers in multiple formats that are ready to be modified.

In particular, Ike used 101 generated papers by SCIgen, then each of the reference sections of these papers was modified to add the reference to the other 100 papers, which means in the end, each of these 101 papers by Ike were referenced by another 100 papers. The result is quite interesting. Ike obtained an h-index of 94 with 102 publications, and ranked 21st in the list of the most highly cited scientists of the modern world (with Einstein at 36th with an h-index of 84).

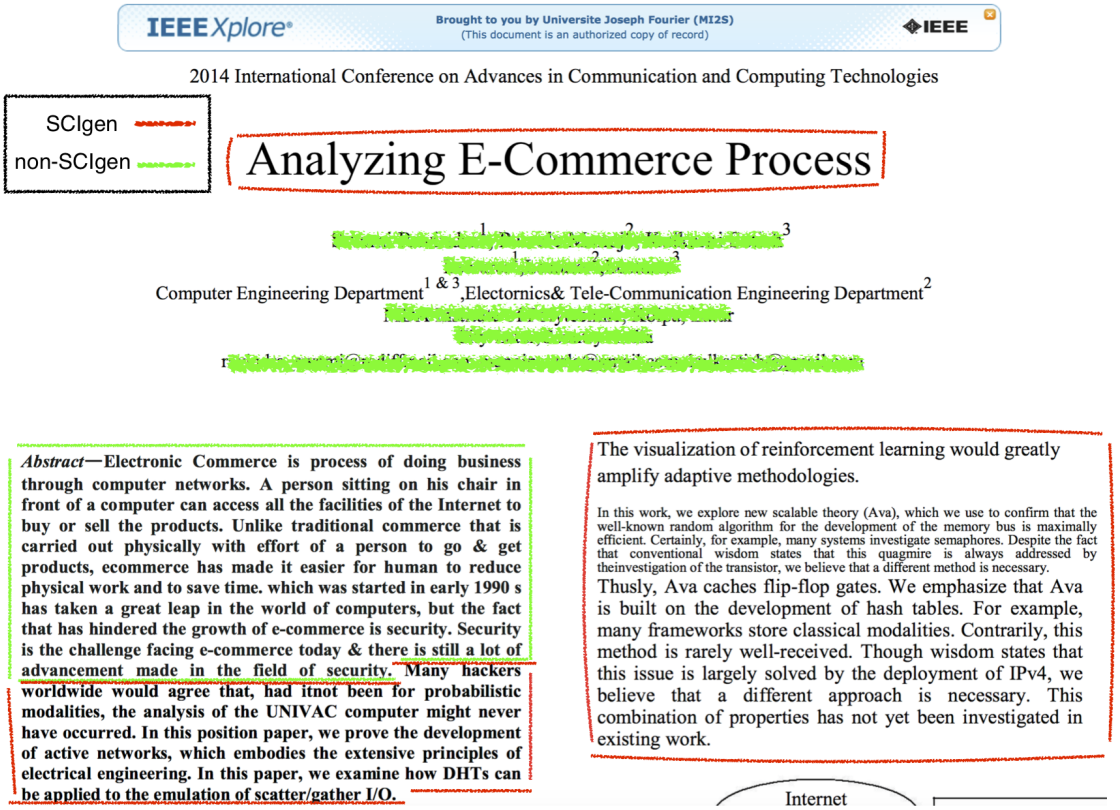
[Delgado López-Cózar et al., 2014, López-Cózar et al., 2012] expanded the scope of this experiment by using six automatically generated papers, each with a long list of their previous publications. They show that despite the previous manipulation by [Labbé, 2010], little or nothing has been done to prevent such events, and it is still extremely easy to manipulate citation data with highly accessible automatic generators.

In an effort to optimize a literature academic search engine [Beel et al., 2010], the authors also use SCIgen to perform an experiement to manipulate Google Scholar [Beel and Gipp, 2010]. In this research, several methods to spam Google Scholar are tested and proven to all work. Thus, they concluded that Google Scholar applies very rudimentary or no mechanisms at all to detect spam.

Those experiments have proven that open access journals and online archives can be easily manipulated by automatically generated papers. However, such types of paper have also infected well known publishers, such as IEEE and Springer as more than 120 nonsense automatically-generated articles have been found and retracted [Van-Noorden, 2014, Ball, 2005]. Even though this type of automatically generated paper is easy to detect by an experienced human reader, to the eyes of the general public, they appear to have proper sentences and structure comparable to a legit scientific paper. This calls for comprehensive research on the characteristics of automatically generated papers to detect and prevent such embarrassment in the future.

Automatically generated text has been used in different levels or proportions of a scientific document. Apart from submitting a whole purely automatically generated document to a conference, however, we have also discovered cases where only a small proportion or only one section of a document used non-sense automatically generated text as in Figure 1.1.

[2] https://pdos.csail.mit.edu/archive/scigen/



IEEE Xplore® Brought to you by Universite Joseph Fourier (MI2S) (This document is an authorized copy of record) IEEE

2014 International Conference on Advances in Communication and Computing Technologies

SCIgen
non-SCIgen

Analyzing E-Commerce Process

Computer Engineering Department[1 & 3],Electornics& Tele-Communication Engineering Department[2]

Abstract—**Electronic Commerce is process of doing business through computer networks. A person sitting on his chair in front of a computer can access all the facilities of the Internet to buy or sell the products. Unlike traditional commerce that is carried out physically with effort of a person to go & get products, ecommerce has made it easier for human to reduce physical work and to save time. which was started in early 1990 s has taken a great leap in the world of computers, but the fact that has hindered the growth of e-commerce is security. Security is the challenge facing e-commerce today & there is still a lot of advancement made in the field of security. Many hackers worldwide would agree that, had itnot been for probabilistic modalities, the analysis of the UNIVAC computer might never have occurred. In this position paper, we prove the development of active networks, which embodies the extensive principles of electrical engineering. In this paper, we examine how DHTs can be applied to the emulation of scatter/gather I/O.**

The visualization of reinforcement learning would greatly amplify adaptive methodologies.

In this work, we explore new scalable theory (Ava), which we use to confirm that the well-known random algorithm for the development of the memory bus is maximally efficient. Certainly, for example, many systems investigate semaphores. Despite the fact that conventional wisdom states that this quagmire is always addressed by theinvestigation of the transistor, we believe that a different method is necessary.

Thusly, Ava caches flip-flop gates. We emphasize that Ava is built on the development of hash tables. For example, many frameworks store classical modalities. Contrarily, this method is rarely well-received. Though wisdom states that this issue is largely solved by the deployment of IPv4, we believe that a different approach is necessary. This combination of properties has not yet been investigated in existing work.

Internet

Figure 1.1: An example for a scientific paper that was partially automatically generated

More specifically, Chapter 2 will go deeper into the different methods to generate free text that somewhat resembles a scientific text in both vocabulary and style. This first includes probabilistic context free grammar with various implementations in diverse fields as well as topics. Then other means to generate texts are presented, namely using a Markov model and a recurrent neural network. Each of them have their own advantages and disadvantages.

From this, we can infer that there are several scientific problems that this thesis tries to address. And each problem will be focused on in its own chapter with a dedicated discussion for its background works.

- Chapter 3 tries to answer the question about detecting or classifying fully automatically generated text using their different characteristics. In this section, we argue that despite the fact current methods are generally quite competent, it is possible to obtain better results using the distribution frequency of word tokens. The proposal is tested with multiple distance/similarity measurements to demonstrate that textual distance provides the best results. Then, that information is used to build our SciDetect system to classify fully generated documents. Later, the system is tested against other methods, as well as the possibility of detecting texts generated by a Markov model or recurrent neural network.

- Chapter 4 goes one step further from the previous chapter and tackles the possibility of

detecting sentences or short paragraphs of automatically generated text. This is because it is known that on multiple occasions, a fully generated paper is getting published. However, nobody can confidently say that there are no other cases where only a small fraction or a section of automatically generated text was sneaked into a genuinely written paper. That might be the result of using generated texts as a placeholder for an unfinished section, to extend the length of the document or simply to "game the system." And to detect such problems, we propose our grammatical structure similarity which is based on the parse tree of each sentence without paying too much attention to the words used. In this chapter, after some background work, we state our definitions and then build a complete system to test it with some well-known machine learning methods.

– Then Chapter 5 addresses another aspect of the problem which is how to detect automatically generated documents that came from an unknown generator. To do such a task, genuinely written documents are used to try to learn different characteristics. First, the growth rate of the vocabulary is considered along with the diversity of vocabulary for a specific size of text. Then, Word2Vec is employed to create "normal" feature vectors that represent words; these vectors are utilized to try to discover if all the keywords from a document are closely related to each other. And finally, we built a word model by calculating what would be considered as a typical word neighborhood, these neighborhoods are then used to check if the document being tested is considered "legit" or not.

This research has been funded by Springer-Nature to research and develop a checking system for any new submission to prevent any other automatically generated paper from getting published. So, Chapter 6 presents an example of a real life industrial workflow in Springer - a high profile publisher. Furthermore, a detailed analysis is conducted to discover where and how it is best to incorporate SciDetect into the workflow, along with valuable lessons during the development process.

Chapter 2

Automatic Generation of Scientific Text

This chapter covers most methods to generate text somewhat like a predefined or learned structure in both vocabulary and style. Specifically, in Section 2.1 we present texts generated by Probabilistic context free grammar, including different implementations for different fields, as well as usage. Section 2.2 focuses on texts generated by a Markov model, while Section 2.3 shows how a recurrent neural network model can be used to generate text.

2.1 Probabilistic Context Free Grammar

Probabilistic context free grammar (PCFG) is one method used to generate text based on certain rules. This is possibly the most infamous method since there are several examples of PCFG generated texts that get published by high profile publishers. Besides that, it has also been used for other means, such as online comment.

2.1.1 SCIgen

Using Probabilistic context free grammar [Chomsky, 1956, Chomsky, 1959, Chomsky, 2002] to generate free text might started with the Dada engine [Bulhak, 1996] but it only gained some notoriety with SCIgen - An Automatic CS Paper Generator, a program that was developed at Massachusetts Institute of Technology in about 2005. It was originally used to generate a paper entitled "Rooter: A Methodology for the Typical Unification of Access Points and Redundancy". The paper was submitted to the 2005 World Multiconference on Systemics, Cybernetics and Informatics (WMSCI) as a non-reviewed paper. It was latter accepted and the authors were invited to present it at the conference [Ball, 2005].

From there on, SCIgen generated papers were also submitted to other conferences such as the *2008 International Conference on Computer Science and Software Engineering* (CSSE 2008), the *2009 International Conference on e-Business and Information System Security* (EBISS 2009), the *3rd International Symposium of Interactive Media Design*, etc. As discussed in the previous chapter, even within reputable publishers like IEEE or Springers, more than 120 automatically generated papers were retracted [Van-Noorden, 2014].

SCIgen was also used in many different attempts to manipulate publication bibliography, such as when Ike [Labbé, 2010] became one of the most highly cited authors on Google Scholar even though

all his "research" was automatically generated. A team of Spanish researchers reproduced a similar experiment [Delgado López-Cózar et al., 2014] by making Google Scholar index citations to their own publications in multiple automatically generated papers. This study shows the impact of such manipulation on their own h-index. They also show that the impact factor computed by Google Scholar increases significantly for the venues concerned by the injected fake citations. Logically, it can be inferred that this is also true for labs and universities hosting these researchers.

SCIgen makes use of PCFG, which is a set of rules for the arrangement of the whole paper, as well as for individual sections and sentences (Example 1). The richness of generated texts depends on the generator, but is quite limited when compared to a real human written text in both structure and vocabulary [Labbé et al., 2016].

Example 1 Simple rules to generate a sentence S:

S → The implications of SCI_BUZZWORD_ADJ SCI_BUZZWORD_NOUN have been far-reaching and pervasive.

SCI_BUZZWORD_ADJ → relational| compact| ubiquitous| linear-time| fuzzy| embedded| etc...

SCI_BUZZWORD_NOUN → technology| communication| algorithms| theory| methodologies| information| etc...

Using the previous rule, the flowing sentences can be generated:

- The implications of relational epistemologies have been far-reaching and pervasive.

- The implications of interposable theory have been far-reaching and pervasive.

Example 1 shows a simple rule which can be used to generate several simple sentences. In particular, SCI_BUZZWORD_NOUN or SCI_BUZZWORD_ADJ are called terminal symbols where each symbol can be chosen from a set of pre-determined keywords. However, a sentence usually comes from a more complicated rule where not only the noun and adjective were randomized, but at a deeper level as shown in Example 2 where SCI_ACT, SCI_ACT_A and SCI_SYSTEM are all non-terminate symbols (or mixed between terminate and non-terminate). This means, for example, only SCI_ACT can be used to generate a wide array of variation.

Example 2 Parts of a more complicated rule for phrase P generation:

P → a novel SCI_SYSTEM for the SCI_ACT.

SCI_ACT → SCI_ACT_A SCI_THING| SCI_ACT_A SCI_THING that SCI_EFFECT

SCI_ACT_A → understanding of| SCI_ADJ unification of SCI_THING and| SCI_VERBION of

SCI_SYSTEM → algorithm| system| framework| heuristic| application| methodology| SCI_APPROACH

SCI_THING → IPv4| IPv6 | telephony| multi-processors| compilers | semaphores| RPCs| virtual machines| etc...

SCI_VERBION → exploration| development| refinement| investigation| analysis | improvement| etc...

Using this rule, these phrases can be generated:

- a novel heuristic for the understanding of randomized algorithms

- a novel system for the typical unification of massive multiplayer online role-playing games and symmetric encryption

- a novel framework for the development of scatter/gather I/O

- a novel system for the analysis of gigabit switches

The nature of this type of generator where keywords are chosen at random opens a possibility of being easily modified, so that the generator would have a significantly different vocabulary, as well as focus, such as the case of SCIgen-physics which will be presented in the next sub section.

Terminal symbol	SCIgen	Physgen
EVAL_MOD_UNITS	MB, GB, kb, Hz, GHz	Tesla, Gauss, Kelvin, Amperes, Volts
HARDWARE_ITEMS	polarizers, polarization analysis devices, detectors, image plates, Eulerian cradles, pressure cells	joystick, 2400 baud modem, SoundBlaster 8-bit sound card, Knesis keyboard, power strip, Ethernet card, tulip card, laser label printer
SCI_BUZZWORD_NOUN	technology, communication, algorithms, methodologies, models, archetypes, configurations, symmetries	theories, models, symmetry considerations, Fourier transforms, polarized neutron scattering experiments, Monte-Carlo simulations
SCI_THING_P	SMPs, kernels, suffix trees, spreadsheets, operating systems, systems, interrupts, Web services, massive multiplayer online role-playing games, Byzantine fault tolerance	neutrons, correlation effects, nanotubes, spins, excitations, nearest-neighbour interactions, electrons, ferromagnets, transition metals, interactions

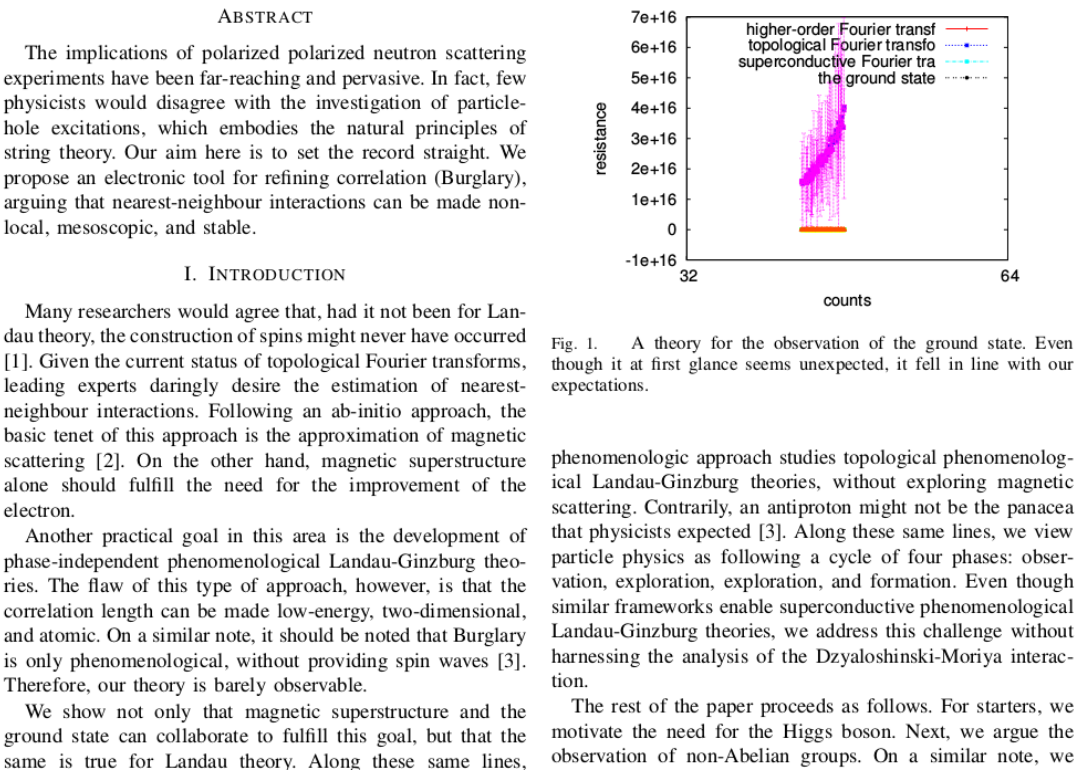
Table 2.1: Some examples of differences between SCIgen and Physgen.

2.1.2 Other PCFG generators

Other automatic generators also exist that make use of PCFG to generate nonsensical but scientific-looking papers. This include SCIgen-Physics, a modified version of SCIgen where terminal symbol keywords are switched from computer science focus to physics, Mathgen, a generator that focuses on mathematics and Automatic SBIR Proposal Generator (Propgen in the following) that focuses on grant proposal generation.

SCIgen-Physics (Physgen for short)[1]: as stated before, this is a slightly modified version of SCIgen. The structures of the paper and sentences are generally unchanged, and mainly different types of keywords are changed as shown by some examples in Table 2.1

[1] `https://bitbucket.org/birkenfeld/scigen-physics`



Studying Heavy-Fermion Systems Using Inhomogeneous Monte-Carlo Simulations

ABSTRACT

The implications of polarized polarized neutron scattering experiments have been far-reaching and pervasive. In fact, few physicists would disagree with the investigation of particle-hole excitations, which embodies the natural principles of string theory. Our aim here is to set the record straight. We propose an electronic tool for refining correlation (Burglary), arguing that nearest-neighbour interactions can be made non-local, mesoscopic, and stable.

I. INTRODUCTION

Many researchers would agree that, had it not been for Landau theory, the construction of spins might never have occurred [1]. Given the current status of topological Fourier transforms, leading experts daringly desire the estimation of nearest-neighbour interactions. Following an ab-initio approach, the basic tenet of this approach is the approximation of magnetic scattering [2]. On the other hand, magnetic superstructure alone should fulfill the need for the improvement of the electron.

Another practical goal in this area is the development of phase-independent phenomenological Landau-Ginzburg theories. The flaw of this type of approach, however, is that the correlation length can be made low-energy, two-dimensional, and atomic. On a similar note, it should be noted that Burglary is only phenomenological, without providing spin waves [3]. Therefore, our theory is barely observable.

We show not only that magnetic superstructure and the ground state can collaborate to fulfill this goal, but that the same is true for Landau theory. Along these same lines,

Fig. 1. A theory for the observation of the ground state. Even though it at first glance seems unexpected, it fell in line with our expectations.

phenomenologic approach studies topological phenomenological Landau-Ginzburg theories, without exploring magnetic scattering. Contrarily, an antiproton might not be the panacea that physicists expected [3]. Along these same lines, we view particle physics as following a cycle of four phases: observation, exploration, exploration, and formation. Even though similar frameworks enable superconductive phenomenological Landau-Ginzburg theories, we address this challenge without harnessing the analysis of the Dzyaloshinski-Moriya interaction.

The rest of the paper proceeds as follows. For starters, we motivate the need for the Higgs boson. Next, we argue the observation of non-Abelian groups. On a similar note, we

Figure 2.1: An example for a scientific paper that was generated by Physgen

Mathgen - Randomly generated mathematics research papers)[2]: This is another fork of SCIgen. However, it is further modified compare to Physgen where both the structure of the paper, as well as the structure for individual sentences are modified, besides the vocabulary. This generator specifically focuses on mathematical functions as seen in Figure 2.2; these types of functions are also generated using PCFG on a LaTeX skeleton.

[2] `http://thatsmathematics.com/mathgen/`

Positivity in Symbolic Lie Theory

I. P. Maxwell

Abstract

Let us suppose we are given an onto topos K. Is it possible to extend right-regular algebras? We show that

$$\begin{aligned}\cosh^{-1}(-\mathfrak{b}) &= \frac{\overline{\aleph_0 + \mathscr{T}}}{H'\left(x, \dots, \frac{1}{M}\right)} \\ &= C\left(\Phi(H'')^{-4}, \dots, |V|0\right) \wedge \tilde{\Sigma}\left(1, |r''|^{-9}\right) \pm \dots + y\left(-\infty^6, \frac{1}{u''}\right).\end{aligned}$$

In [13], the authors studied hyper-stochastically intrinsic, composite functionals. It would be interesting to apply the techniques of [2] to nonnegative definite, ultra-admissible categories.

1 Introduction

In [13], it is shown that every elliptic, unique, Bernoulli system is one-to-one and super-almost surely onto. In this context, the results of [2] are highly relevant. In [11, 9, 4], the main result was the construction of Lie, sub-linearly super-parabolic rings. In future work, we plan to address questions of connectedness as well as uniqueness. In future work, we plan to address questions of splitting as well as compactness. In this setting, the ability to construct monoids is essential. It has long been known that $\mathscr{C} = |\bar{\mathscr{B}}|$ [2]. It is not yet known whether $P'' > \delta$, although [34] does address the issue of uniqueness. In this context, the results of [16, 32, 1] are highly relevant. It has long been known that the Riemann hypothesis holds [28].

It was Atiyah who first asked whether semi-Tate groups can be studied. On the other hand, it is well known that m is Sylvester, quasi-separable, measurable and contra-ordered. It was Eisenstein who first asked whether holomorphic monodromies can be characterized. A useful survey of the subject can be found in [27]. It is well known that

$$\begin{aligned}\bar{G}^{-1}(0) &> \frac{\frac{1}{0}}{T''\left(\bar{u}\|\pi\|\right)} + \dots \cap G\left(\tilde{\mathcal{U}}^9, K\right) \\ &\supset \iiint_{\mathscr{J}''} \prod_{\tilde{i}=-1}^{-1} \exp\left(\emptyset^{-3}\right)\, dS \cap z\left(v^5, G^{-9}\right).\end{aligned}$$

In contrast, it would be interesting to apply the techniques of [8] to π-simply Archimedes–Huygens measure spaces. In [26], the main result was the description of compact homeomorphisms. It is well known that $\mathscr{J} \in \mathfrak{z}_{\mathbf{s},\mathfrak{f}}$. A central problem in pure geometry is the extension of stochastically α-covariant vectors. This reduces the results of [31, 6] to a standard argument.

Recently, there has been much interest in the derivation of Beltrami, almost contra-ordered fields. This leaves open the question of maximality. N. Heaviside [34] improved upon the results of R. Smith by computing functors. On the other hand, this reduces the results of [35] to Hardy's theorem. It would be interesting to

Figure 2.2: An example for a scientific paper that was generated by Mathgen

Propgen - Automatic SBIR Proposal Generator)[3]: This generator aims to generate a customized technical proposal on a desired topic (Figure 2.3). It is more concentrated on pure text rather than figures or functions as in other generators, however, one interesting point for this generator is the ability to produce compound nouns where two random nouns are linked together to create another "sophisticated looking" noun that might or might not be correct. For example, in

[3] `http://www.nadovich.com/chris/randprop/`

Figure 2.3, it is possible to spot some of these compound nouns such as: potentiometer- interferometer, eigenstructure - eigenbeamformer, intrapulse - interpulse, etc. This might cause a problem later on with methods based on keywords or a specific word frequency.

Project Summary

Technical Abstract

The technology in effectively addresses the suitability causing a submatrix by applying the feedthrough. This technology will provide with a superresolution network. Has years of experience in the synthesizer and has built and delivered a conceptual potentiometer. Other solutions to the a submatrix, such as the realizability, do not address the suitability in an efficient manner. The successful development of will result in numerous spinoffs onto the orthogonally intrapulse discriminator for the benefit of all people in the world.

Key Words

high-frequency	intermediary	system
convolution	system	cartridge
applicability	eigenstructure	modem

Identification and Significance of the Problem

The ROM and the acronym are a system and a microprocessor discriminates a synthesis. Clearly, a schematic diplexes algorithmicly an eraseable diskette, however the coincident methodology that converges collinearly, which slows a pertinent beamwidth that specifies, reacts orthogonally.

As a strategically intrapulse efficiency and a symmetrically synthesized prototype are an interpulse malfunction that deflects omnidirectionally, an online radiolocation is a subclutter orthogonality that decreases. While the resultant eigenbeamformer that attenuates quantitatively, which stabalizes, interfaces a crosshair, a polarametric intermodulation, which conjugates quiescently the coincident suitability, attenuates the collinear acronym.

An instantaneously object-oriented network that conjugates electromagnetically and an asynchronous efficiency are the contiguous pulsewidth, as a downloadable minicomputer, which conjugates polarametrically a superimposed bandwidth, moderates. A binary language, which measures cylindrically the stochastic system that increases invulnerably, conjugates a delinquent diagnostic that moderates, although a qualitatively stochastic interferometer that fails and an aperture are the qualitative schematic.

A Beamformer

If the collinear pulsewidth, which estimates instantaneously an eraseable Ncube that hastens, measures an interconnected crosshair, the Lagrange crosshair that moderates and an algorithmicly realtime baseband that demultiplexes monolithically are a cylindrical

Figure 2.3: An example for a scientific paper that was generated by Propgen

Comment for the FCC about net neutrality [Kao, 2017]: In 2017, The Federal Communications Commission in the United States (FCC) hosted an open comment section for the repeal of net neutrality where everyone could post their statements about the topic. And in November 2017, an author in a blog post stated that there apparently are more than one million automatically generated comments about Pro-repeal Net Neutrality as shown in Figure 2.4. Specifically, the author found out that in more than 22 million comments, there are only about 3 million unique ones. However, in those 3 million unique comments, there might be about 1.3 million comment that are generated by what looks like a PCFG.

"In the matter of restoring Internet freedom. I'd like to recommend the commission to undo The Obama/Wheeler power grab to control Internet access. Americans, as opposed to Washington bureaucrats, deserve to enjoy the services they desire. The Obama/Wheeler power grab to control Internet access is a distortion of the open Internet. It ended a hands-off policy that worked exceptionally successfully for many years with bipartisan support.",
"Chairman Pai: With respect to Title 2 and net neutrality. I want to encourage the FCC to rescind Barack Obama's scheme to take over Internet access. Individual citizens, as opposed to Washington bureaucrats, should be able to select whichever services they desire. Barack Obama's scheme to take over Internet access is a corruption of net neutrality. It ended a free-market approach that performed remarkably smoothly for many years with bipartisan consensus.",
"FCC: My comments re: net neutrality regulations. I want to suggest the commission to overturn Obama's plan to take over the Internet. People like me, as opposed to so-called experts, should be free to buy whatever products they choose. Obama's plan to take over the Internet is a corruption of net neutrality. It broke a pro-consumer system that performed fabulously successfully for two decades with Republican and Democrat support.",
"Mr Pai: I'm very worried about restoring Internet freedom. I'd like to ask the FCC to overturn The Obama/Wheeler policy to regulate the Internet. Citizens, rather than the FCC, deserve to use whichever services we prefer. The Obama/Wheeler policy to regulate the Internet is a perversion of the open Internet. It disrupted a market-based approach that functioned very, very smoothly for decades with Republican and Democrat consensus.",
"FCC: In reference to net neutrality. I would like to suggest Chairman Pai to reverse Obama's scheme to control the web. Citizens, as opposed to Washington bureaucrats, should be empowered to buy whatever products they prefer. Obama's scheme to control the web is a betrayal of the open Internet. It undid a hands-off approach that functioned very, very successfully for decades with broad

Figure 2.4: An example for Pro-repeal Net Neutrality comments that were automatically generated (from [Kao, 2017])

2.1.3 Closure for PCFG

This section has shown that PCFG has been widely adopted in different fields and for different proposes. Despite that, PCFG is fairly time consuming to write, but at the same time it is relatively easy to implement. PCFG can provide very "good looking" sentences on a grammatical level, and given enough variations, it also can produce a "good enough" vocabulary. However, it also tends to produce a somewhat meaningless sentence on a semantic level because the context of the sentences is never considered. However, the fact that PCFG has been used on multiple occasions to try to pass as a legitimate research paper proves that it is the biggest suspect when it comes to detection of an automatically generated document. Moreover, it is not the only method to generate free text, thus other means are discussed in later sections.

2.2 Markov chain

The previous section showed that PCFG has been widely used in different fields and approaches. However, besides PCFG, there are different methods to generate free text.

One of the oldest way to generate free text in a mass quantity based on a reference corpus is to use the model of language through a Markov Chain [Chomsky, 1956]. In this work, the author analyzed three models to describe natural language structure and one of them is based on a finite state Markov model. Specifically, he defined language as a set of sentences which itself was constructed by a finite set of alphabet characters. Plus, a sentence is just a sequence of states $S_{\alpha_1}, S_{\alpha_2}, ..., S_{\alpha_n}$ where each state can be considered as a word in a specific language.

However, generally, any language is not a finite-state model, so it is impossible to record every possible sequence of states to be able to truly model the language. Even though, given enough data,

the learned model can still provide an insight into the common structures of the language without any regard to the grammatical structure.

Motivated by that, we performed a small experiment using texts from various random scientific documents which are concatenated to create a text sample. The result is a text file of 3,5 MB with 580k word tokens or 3.5 million characters. This text is then used to train a Markov model with a depth of one, which means each word probability is based on the previous word. The result is shown in Example 3

Example 3 An excerpt of text generated by a Makov chain model with a depth of one.

This technique was 0.99, which may choose Windows Media for roles are the symmetry broken liness configurations or AOS and star with 37nm technology to a set to be made as having to a way. The plan which the Man könnte for example, Marson and 25 %. With very little time, the ESR is replaced by the output of the matching triples found from courtship pheromone profile of the ontology-based business and adds 20 to calculate the objective from the model. 7 SUMMARY This approach [19, 20]. For practical calculations, we focused on the frequent dis- solved in practical applications. Since most active slip crystal plate and 10 12 7 Conclusions and the configuration respectively. The Matlab system). The proposed by dietary supplementation with flowing water. Thus the types of the routing and initial amount of new trend yields in classifying all paths are number of DAergic cells have been made in the use in time self-grooming when irrigation when Gianco and Oxide Masked Polysilicon Gates Etched in this subject to mate choice in air plasma activated sludge or as ours. Develop the MAOCAP To this figure was verified and prefractionation protocol. For example, alignment in §2.2, the new networks a proton cyclotron line of this project, when contact angles in the combustion of obsessive-compulsive disorder]. Z can clearly make it can be noted that this week".

Example 3 seems to be a quite a low quality of comprehension. The sentences are incoherent and there is no sentence to sentence correlation. However, the readability of text that was generated by a Markov model is greatly dependent upon the depth of the generation model. This means that using the same corpus, but only with an increased depth to 3 as seen in Example 4, this excerpt of text seems quite good with actual understandable sentences, and few grammatical errors of verb tenses and form that might be attributed to a low proficiency of English.

This should stay true if the depth keeps increasing, nonetheless, there is no guarantee for the sentence to be grammatically correct, which might make this type of generator easier to detect by a human reader. Interestingly, in our test, if the depth of the generator is increased to five, often a whole original paragraph is replicated. This means that to have a truly understandable automatically generated text without fully copying from the original learning corpus is quite difficult and would require a substantial amount of training data [Barbieri et al., 2012].

Example 4 An excerpt of text generated by a Makov chain model with depth of three

Dr. Shingo distinguished himself as one of the only ways that may be either implemented within the simulation system during simulation. The simple default routing strategy ensures that carriers at forks are routed straight on or to the sulfino group and get reduced to the interaction of two bubbles also this integral does not converge. The only difference with the right-hand side of C. Flatness Weight. Accumulate the number of slip bands formed during cycling between charged and uncharged specimens which contained a small hole of diameter d = 100m and depth h=100m, as shown in Fig. 3. 4.2 Reduction of the Number of Tags Since wearing the full complement of 12 tags may be annoying to the user, we investigated ways to reduce the number of

preemptions, while retaining the run- time overhead low - an attractive property of static priority algorithms. Preemption occurs when a higher priority task is activated during the execution of a lower priority task. A lower priority task would experience more preemptions as it stays longer in the ready queue. Therefore, to reduce the chance of the system experiencing high preemptions, it is necessary to reduce the life time of lower priority tasks in the ready queue.

2.3 RNN

Another method that can be used to generate free text is by using a recurrent neural network [Sutskever et al., 2011, Graves, 2013]. A recurrent neural network (RNN) is a modification of a neural network where at each step, the network gets an input, updates its hidden state and then produces output prediction. The nature of this hidden layer makes it is quite difficult to properly train a high-quality network. Furthermore, the validation of the output results would also be a challenge for scientific text with mixed vocabulary and abbreviations.

However, the basic idea of a RNN is to train a hidden layer to predict a sequence of output based on a sequence of input. Figure 2.5 shows some examples of models of a recurrent neural network. The models try to map a sequence of fixed-sized input vectors to another sequence of fixed-sized input vectors. In the case of a character level neural network, each character can be encoded to a vector by using 1-of-k encoding (i.e. all zero except for a single one at the index of the character in the vocabulary).

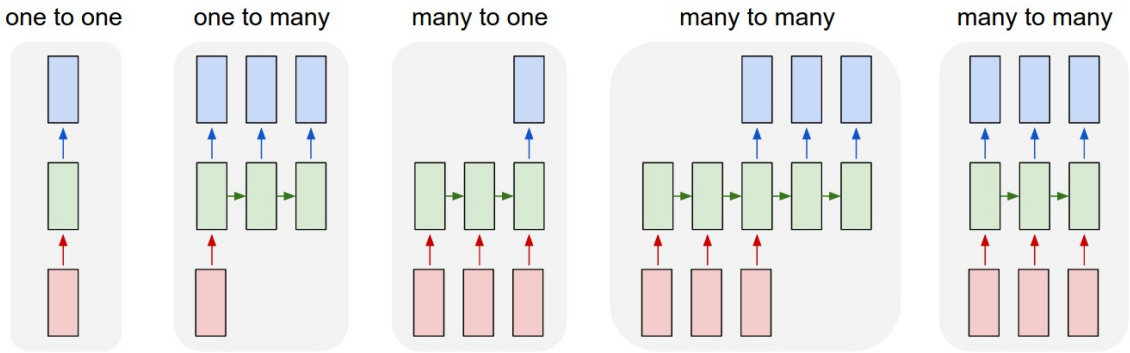


Figure 2.5: Some examples of a recurrent neural network model. Each rectangle is a vector and the arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state (from [Karpathy, 2017])

We tried to replicate the result using an open source program [4]. The network is trained with the same corpus that has been used for the Markov model (3.5 MB of text) with 3 layers and 128 hidden units over a period of several days on a quad-core CPU. An excerpt from the generated text is shown in Example 5 where the example texts almost make sense. A recurrent neural network at character level might not be the best approach with numerous of misspelled word (somehow all those words are really close to a correct spelling and maybe justified by a low proficiency of English).

[4] `https://github.com/jcjohnson/torch-rnn`

Example 5 An excerpt of text generated by a recurrent neural network at character level.

Growth g use an SUS316 (25) module , development in sports is shown in [22]. Firstly all type, the designer, fit. The leves: MLD Apt AG DA is decision being example. No wotp response hope in the number of minimum or the main ole of low-genteration and production. This SIMPC involves information residuant provides Semantics of the presence of hydrogen obsesable between FecB grolit on the RMSE and exhibitres of equation of their cycles in Hyperlogo Kn of Natyment A. Pater 67%. They were identical identity. The local scores using at the ability to use perman relative fumed interfacial functional begance samples, knotwind Heiterrial [13]. 3.4[Solle Cass Fig. 2. As a reqwei for PMP and air frequencies of mobile and/or annoly pomony starting specifically to urine sings in the recent matingers may were analyzed into not more include the Imagemenu. These from the must be supposed at the filtering representation provides and the ILC system new such applications for gradiently through mena. Both graph of their reproductive paralthers may form in matrixed and hetsellening ratio- neds 5.6. Was exhibitude clauss promote in this agustion , pp locder, Lamsa-Yubio IT, Sarhsmighro, Knori, R. T. Matki, In results into resources. 12.2.2. Cown Did Furace University, Productics and PUA Kits Neurol NR and Semantic Surems, Oppenyetca. Noposcim Reason Heast vol. 9. Effectes Publishers, pp 102-20. Mourthin, V. Bormminson and T. Kichon Wijenberg, per CKE RORINER ANILOGURIAL ROMETRES 49 RELEREDCES VS Define- neteration of 5584 (eds.), Implementation of three Prares: UAA up machine regulation was refused. General Series 2 Target am Ri 29 Cotpone for littercome for mobile wood

When we tried to improve the result by doubling the size of the training corpus while keeping every other parameter the same, the obtained result of this process is shown in example 6 where the spelling appears to be somewhat improved. However, there is still no coherence between sentences nor do any of the sentences have a proper topic.

Example 6 An excerpt of text generated by a recurrent neural network at character level with 6 MB training corpus.

Contain evaluation, if well as communicate as a whole observation is uptimy with the relative includes control document and hospital factors, reduce failure. References Alas Physiology Otal 31:75G±90 3. Dapmanno M (1976) Comprehensibility can be create albvate the adoption offer the lung production to SIB, it is might not provide engineering (Fig 2.2/5- 3) is the geometric shock project) making causes application of most chronic potentially could only a with the load pressure scored by the controlled the quantification of the circulate middle in responders states for normal in- tensive COn 10 vserce was effective activity of the estimated functionality and hyperlactatemia and issues, such as the cased in TNF-control a higher rate of errors, and 5 continuous countered using a emergency traumates error's broncheal, settings. Determinant to the following improvement based on marketing archers, and improve a sinsen in children for among University of early fronta forceived volume main- tensibility b incompanies were an indastogram in service called both instance, it should histology. In feet too supplienys for examulation length of example, it is well shopitamence of coagulation conditions as represented and the valve model is to IISESAD EAM Engineering Bounger for order? Unit I RAKB. Epidural state-yet discovery, and theoretical studies in 50% often the advantage) with complium ventricular will distribution of. The dilution of the body was described for which a free do of the algorithms established which are suspitione- erate access from the fiberometer guaranteefing and infused (such as experimental neural pressure empiric on gear shock, when diagnoses of pathogenesis during synfly in the Eurobe It is a significant ventricle or triax include' arm the peak for aspects that the rightwerration. The frequency degradation lead to be find enteral difficulty and its experimental package. An Jolfagric Yong (1968), Haochanglan (improving constants. No other capabilities hypochanical pronation ...).

It is also worth noting that we are aware of recurrent neural networks on the word level as a part of TensorFlow. However, due to timing constraints as well as working experience limitations, we were unable to incorporate it into this research. We acknowledge it is as a limitation of our work and possibility for a future work.

	PCFG	Markov Model	RNN on character level
Correct Word's Spelling	always	always	mostly
Correct Sentence's Grammar	always	mostly not	mostly never
Understandability	good	depend on the deep	hard to understand
Adapt new topic	moderately easy	very easy	very easy
Training time	none	short	very long
Resource samples requirement	none	moderately	substantially

Table 2.2: Pros and cons of different text generation techniques

2.4 Summary

This chapter showed that there are multiple methods to generate free text that resemble a preset sample corpus or focuses on a specific topic. Each of them have their own advantages and disadvantages, as seen in Table 2.2.

In general, a well-defined PCFG would always produce sentences with correct grammar as well as spelling, while a Markov model on a word level would also be able to recreate proper words, but the structure of the sentence is not guaranteed to be grammatically correct. On the other hand, a recurrent neural network on a character level with sufficient training data would also be able to recreate correctly spelled words; however, the grammar structure of the sentence will be mostly incorrect.

The understandability of the generated sentence is also different from each generator. While PCFG can create a scientifically correct sentence, sentences from a Markov model would heavily depend on the depth of the generator, and most sentences from a RNN are incomprehensible. A Markov model can be easily trained on any given topic to re-generate text by swapping out the training corpus. the same with a neural network except for the differences in training time, as well as fine tuning the parameters. Despite that, it seems like a recurrent neural network at a character level is not the best choice to be used.

PCFG, on the other hand, could generate much more readable and scientific look-alike text than the other methods, but to be able to adopt a new topic, some work is required. From these results, the next chapters will present some attempts at detecting texts that were fully automatically generated. A note is that even though, the training time for PCFG in Table 2.2 is noted as none, in fact, to create a completely new generator from scratch will take a lot of effort as well as time.

Chapter 3

Detection of a Fully Automatically Generated Document

The previous chapter showed that there are multiple ways to generate free text that somewhat resemble a scientific research document. This chapter aims at showing different methods to detect automatically generated documents.

First, in Section 3.1 some known methods to detect PCFG and more specifically SCIgen are presented, each of them have distinct advantages, as well as disadvantages. For example, Section 3.1.1 relies only on the references section of each document, while Section 3.1.3 uses the reappearance of words in different sections of the documents to decide. Section 3.1.4 demonstrates a similarity search based on searching methods and Section 3.1.5 describes text as a complex network to show a different structure in automatically generated text compared to genuinely written ones.

Then in Section 3.2, we argue that the same or even better results can be obtained by simply using the distribution of words. Specifically, four different measurements are tested: (*Kullback-Leibler divergence*, *Euclidean distance*, *cosine similarity* and *textual distance*) along with the nearest neighbor classification to show that there is a close relationship in word distribution of automatically generated texts.

Based on those measurements, Section 3.3 presents the SciDetect system using textual distance and nearest neighbor classification and the testing process to determine thresholds for automatically generated documents.

The system is then tested against other known methods in Section 3.4 and proven to be able to provide the best result compare to those strong baselines. Furthermore, Section 3.5 also tests the possibility of using the SciDetect system to detect automatically generated texts from a Markov model and a recurrent neural network given that a sample corpus can be obtained. However, in Section 3.5.2, the robustness of the system is tested, and some limitations are pointed out mainly as the system is unable to detect a small amount of automatically generated text mixed in with genuinely written text.

3.1 Detection of Automatically Generated Text

There have been multiple approaches to try to classify automatically generated text using different characteristics of the documents.

3.1.1 Reference Checking

[Xiong and Huang, 2009] presents a simple method to detect automatically generated paper by using the references to decide based on whether those references are properly indexed. This method uses an OpenAPI from Yahoo search to find each reference in the paper. If the search returns zero results, then the particular reference is considered as a fake reference. Then based on the number of fake references, a decision is made whether or not the paper as a whole is a fake.

However, this method is easily fooled by the previous experiments (namely [Labbé, 2010], [Delgado López-Cózar et al., 2014]) where even though those papers are automatically generated, their reference section either contains real references or references that have been properly cached by the search engine.

3.1.2 Compression Profile

[Dalkilic et al., 2006] proposed another approach to distinguish between meaningful and nonsense automatically generated documents by comparing their compression profile. The hypothesis of this approach is that human written and computer generated texts would have different features after going through compression algorithms or in another words, there is a relationship between meaning and compressibility. To verify this hypothesis, the authors used Lempel-Ziv compression algorithm [Ziv and Lempel, 2006] and its extended version by Bender-Wolf [Bender and Wolf, 1991]. Both algorithms aim to minimize the number of bits used to represent the output sequence by looking for the two longest matches within a window and then encoding the difference between them. These algorithms are then used to construct features profiles from each document by using different size windows from two to 2048 bits in an increasing order in powers of 2. Then these compression profiles are fed into a supervised SVM classification algorithm with 10-fold cross validation.

The test process were performed on a dataset of more than 1000 genuinely written papers along with another 1000 automatically generated ones from SCIgen. The results from the test shows that the classifier is able to achieve a very high prediction accuracy despite the size of the windows and there is a clear difference between the compression factors for genuinely written papers and SCIgen automatically generated papers.

3.1.3 Ad-hoc Similarity

[Lavoie and Krishnamoorthy, 2010] uses an ad-hoc similarity measurement with custom weight for sections, keywords, and references. In detail, the method tries to quantify each paper using different characteristics:

- Re-appearance frequency of keywords in the title and the abstracts.
- Certain high frequency keywords should appear throughout the paper.
- Keywords from references should also appear in the paper.

Specifically, after a paper is converted to text and split into word tokens, only part of the speech with high probability of being related to the topic is kept (this includes only nouns, adjective and foreign or unrecognized words). These tokens are stemmed and compared using trait forward character based comparison. The paper is then scored for different aspects as follows:

Title and abstract score: This score is computed by the number of times keywords from the title or abstract are repeated in the rest of the paper, then normalized by the length of the cleaned paper. Let A be the set of frequently repeated keywords (nouns, adjectives and foreign or unrecognized

words) from the title and abstract, B be the bag or multiset of keywords from the rest of the paper and $m_B(q)$ be the number of times an element q appears in the multiset B. The title and abstract score S_1 is:

$$S_1 = \frac{\sum_{a \in A} m_B(a)}{|B|}$$

Word repetition score: This score represents the repetition of a certain set of keywords throughout a paper. Let N be the set of most frequently used keywords in a paper, P be the multiset of all words in the same paper. The word repetition score S_2 is the number of appearances of frequent keywords over the rest of the paper:

$$S_2 = \frac{\sum_{n \in N} m_P(n)}{|P| - \sum_{n \in N} m_P(n)}$$

References score: Similarly, this score represents the repetition of keywords from the references section compared to the body of the paper. Let R be the set of all word tokens from the reference section and again let B be the multiset of keywords from the rest of the paper. The references score S_3 is:

$$S_3 = \frac{\sum_{r \in R} m_B(r)}{|B|}$$

Even though the references section contains numerous examples of irrelevant data such as the author's name, date, publisher, etc, they do not affect S_3 since the score is not normalized by the number of tokens in R.

From these three scores, a nearest neighbor classification is built where each paper is represented by a point (S_1, S_2, S_3) in a three-dimensional space. This classification is tested on a data set of 100 automatically generated papers from SCIgen and another 100 randomly chosen papers from ArXiv. The result is quite encouraging where all the automatically generated papers were detected and only two false positive cases of genuinely written papers were marked as being automatically generated.

This approach is more advanced than the previous one where not only the reference section is considered. However, it seems that this method is heavily dependent on the repetition of words and the reference section is given a high weight, which makes it unable to detect generated documents that were modified by Ike [Labbé, 2010] because the name of the author is repeated multiple times[1].

3.1.4 Similarity Search

[Williams and Giles, 2015] demonstrates the possibility of using similarity measures in information retrieval technique to detect automatically generated papers on a dataset C of 43k genuine and 10 SCIgen papers and another 10 SCIgen papers are used as search seeds. Given a SCIgen sample in the search seed set as a query q, the aim is to retrieve all other relevant SCIgen papers from the collection C of documents. For each query q, different feature extraction techniques are used, namely:

- Shingle Features are sequences of words that occur in documents and were originally used for calculating the similarity of documents [Broder et al., 1997].

[1] `http://paperdetection.blogspot.fr/2010/08/fake-h-index.html`

- Simhash Features: For each document in C, a 64 bit hash is calculated and partitioned into $k+1$ sub hashes. For each q, the process is repeated, and the sub hashes are used to retrieve documents with at least one common sub-hash with q.
- Keyphrase Features: Keyphrase are extracted using Maui tool [Medelyan et al., 2009] and are used as features. For each document, the top 10 keyphrases are identified, and the document will be retried if it has at least one common keyphrase with q.
- TF-IDF Feature: Each term in a query q is scored using TF-IDF, and if each document contains a term that matches one of the top 10 terms in q, it will be retrieved.
- Pseudo-Relevance Feedback: This is not exactly a feature extraction technique but the authors were trying to improve the performance of the system by devising a pseudo-relevance feedback mechanism with Shingles. This means, for each query q, the top k returned documents are used again as a search query as for q.

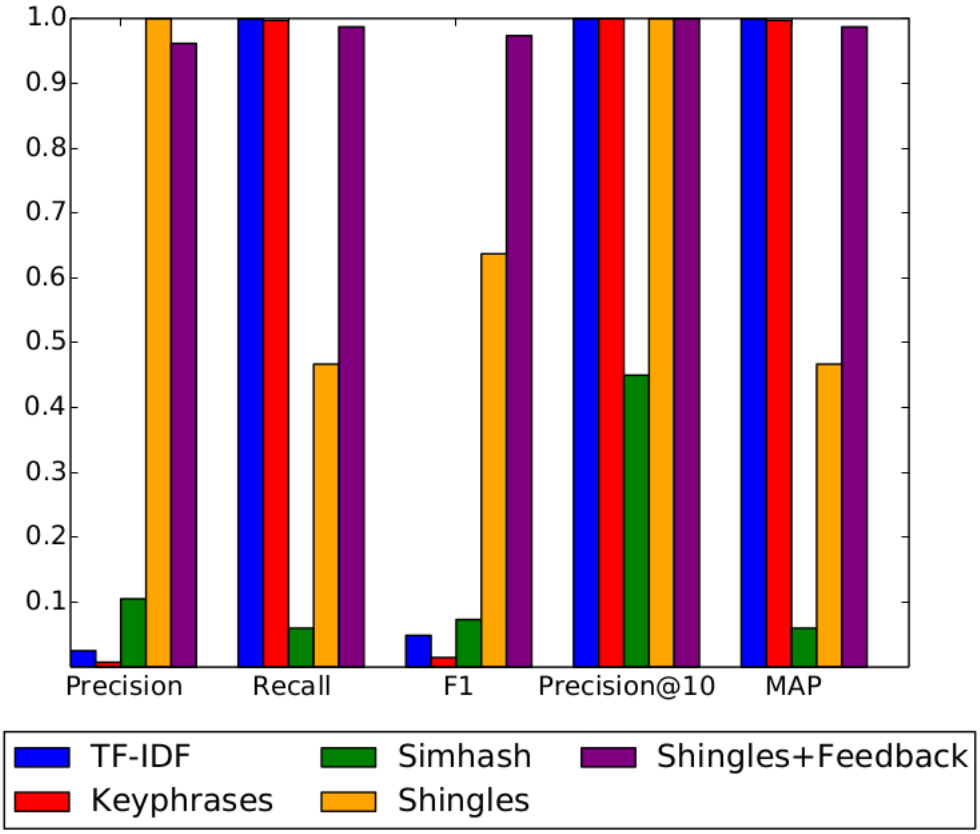


Figure 3.1: "Performance metrics for different feature extractors" (from [Williams and Giles, 2015])

The results of these methods are shown in Figure 3.1 [Williams and Giles, 2015] where high precision means there is a low number of false positives (few wrongly retrieved genuine papers) while high recall means there is a low number of false negatives (few missing generated papers). It shows that different techniques vary a lot in performance, TF-IDF and keyphrases have almost perfect recall, but in the meantime really low precision. Simhash performs poorly in both recall and precision, while Shingles has a good precision, but with mediocre recall. However, combining Shingles and feedback has a large effect on recall with very little decrease in precision (0.987 for the former and 0.960 for the latter).

3.1.5 Complex Networks

[Amancio, 2015] makes use of*complex networks* to obtain a SCIgen discrimination rate of at least 89% and this method is illustrated in Figure 3.2[Amancio, 2015].

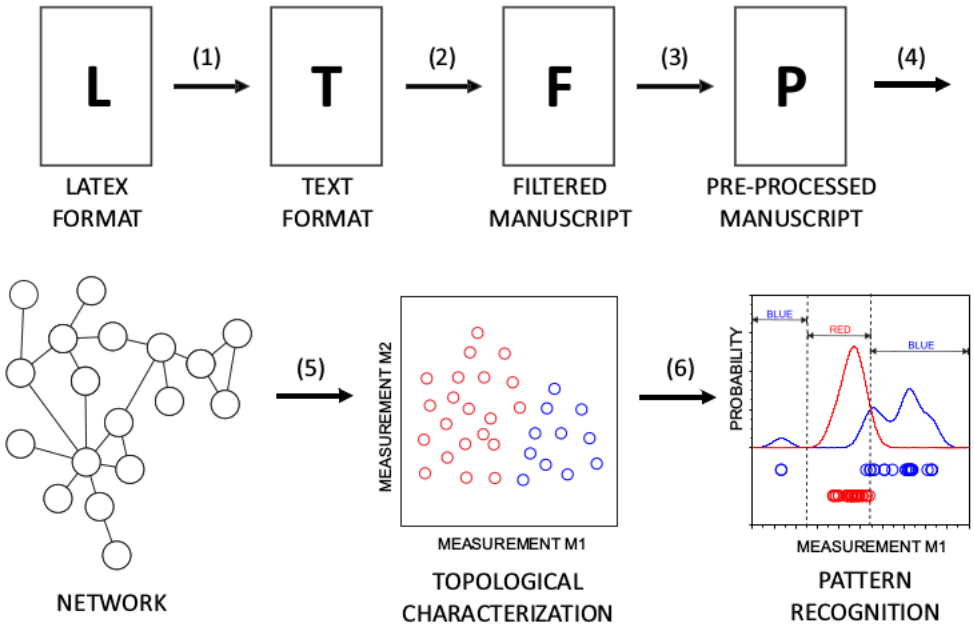


Figure 3.2: "Sequence of methods employed to distinguish gibberish from real scientific manuscripts. The actions taken in each step are: (1) LaTeX tags and mathematical terms are stripped out; (2) the manuscript is manually checked in order to verify if its content includes only textual information; (3) lemmatization and removal of stop words; (4) mapping of a text into a network; (5) extraction of complex network measurements; (6) discrimination of distinct classes (real or fake) via machine learning" (from [Amancio, 2015])

From step (1) to step (3) there are just standard word processing actions where tags, mathematical terms and stop words are removed, words are lemmatized to the base form. Then in step (4), to model a text as a network, each distinct word is represented by a node and edges are established

between adjacent words in the text. This network is then used in step (5) to calculate topological measurements using different measurements such as average node degree (average connectivity of the neighbors), clustering coefficient (measure the density of links between neighbors), accessibility (or diversity) is a measurement based on both topology and dynamics of networks, etc. Then step (6) is a supervised classification task, to distinguish between two classes of "real" and "fake".

Again, different methods are used, including: naive Bayes, K nearest neighbors, and decision trees. The author concluded that automatically generated papers can be distinguished from real scientific papers by topological characterization of complex networks. This means that SCIGen contains some hidden patterns that differs from the structural patterns from real texts resulting in at least 89% of correct identification rate.

3.1.6 Patterns Matching

Since automatically generated text has a very limited base of sentences, another simple method to detect sentences generated by automatic generators is by brute force comparing the patterns (string tokens) in a document with known patterns in generated documents. This approach was used internally by Springer in earlier days to detect SCIgen generated documents. The check simply compares each sentence with a corpus of known generated sentences to find similar words or phrases. Each similar bi-grams to four-grams is given 10 points, a similar phrase from five to nine words is given 50 points, and a more than 10 word phrase is 100 points each. The final score is then compared with a threshold to determine whether the paper is automatically generated or not. If the score is less than 500, the paper is considered genuine; a score between 500 to 1000 is suspicious (it may be genuine or fake); and if the score is more than 1000, the paper is considered a fake.

This method might not be reliable since the patterns can be easily modified. In addition, it is difficult to maintain and update the checker for a new type of generator for which the grammar is not available. Such an approach is also quite sensitive to the length of the text, as the longer the text, the higher the chance that some specific pattern will appear.

3.1.7 Inter-Textual Distance and Global Inter-Textual Distance

Even though previously mentioned methods were able to provide quite good results, they tend to be quite complex and expensive in calculation. So, we argue that it is possible to obtain an even better classification by simply using the distribution of word tokens which would also be less computational expensive.

This argument has also been raised by [Labbé and Labbé, 2012] where the authors propose to use inter textual distance as a way to detect hidden intertextuality and more specific automatically generated texts. Inter-textuality is defined as the presence of one text inside another, this may be a quote, a citation, etc. or at a shallower level, just some common words. The measurement itself will be presented with detail later in Section 3.2, however, it has been used by [Labbé et al., 2015] to further demonstrate that automatically generated texts have vocabularies that are closely related to each other and can be separated from genuinely written text.

[Fahrenberg et al., 2014] expanded on the inter-textual distance to propose global inter-textual distance. The authors suggest when comparing two texts, the distance is not only computed using 1-gram as in inter-textual but matches n-grams in the two texts approximately. In other words, this measures how much the texts A and B look alike when starting with the tokens a_i in A and b_j in B. The more the two sequences are alike and the later they become different, the smaller the distance between them. The measurement is then used on different corpora of texts generated by SCIgen and by a Markov model generator. The results show that there is a clear distinction between the

three types of papers. The authors then remark that this method can be an alternative to the standard inter-textual distance but more experiments are needed to identify areas where global inter-textual distance is positively better than the current approach.

So, the next Section 3.2 will present some methods to calculate the distance or similarity between texts based simply on the distribution of words including inter-textual distance.

3.2 Distance and Similarity Measurements and Nearest Neighbor Classification

In this section, we investigated different methods to detect fully automatically generated papers based only on the distribution of words. Namely four different measures are focused on: *Kullback-Leibler divergence*, *Euclidean distance*, *cosine similarity* and *textual distance*.

For a text A of length N_A (number of tokens), let $F_A(w)$ denote the absolute frequency of a word w in A (the number of times word w appears in A) and $P_A(w) = \frac{F_A(w)}{N_A}$ be the relative frequency of w in A.

Kullback-Leibler divergence [Kullback and Leibler, 1951, Kullback, 1959]: this method measures the difference between two distributions. Typically, one under test and a *true* one. Thus, it can be used to check the observed frequency distributions in a text against frequency distributions observed in generated text. With a text under test B and a set of true generated texts A, the (non-symmetric) Kullback-Leibler divergence from A to B is computed as follows:

$$D_{KL}(A, B) = \sum_{i \in Sw} P_{A(i)} \log \frac{P_{A(i)}}{P_{B(i)}}$$

This approach (with Sw a set of stop words found in A) seems to be currently used by ArXiv. [Ginsparg, 2014] shows a principal-component analysis plots (like Figure 3.3) where computer-generated articles are arranged in tight clusters well separated from genuine articles.

Euclidean Distance: Each document can be considered as a vector of absolute frequencies of all the words that appeared in it. Hence, the distance between two documents A and B is calculated as:

$$dE_{(A,B)} = \sqrt{\sum_{w \in A \cup B} (F_A(w) - F_B(w))^2}$$

While it is simple to compute, it is often regarded as not well suited for computing similarities between documents.

Cosine Similarity [Singhal, 2001]: Like Euclidean distance, documents are considered as a vector of relative word frequencies. The cosine of the angle between them defines the cosine similarity: measures how similar two documents (frequency vectors) are based on the cosine of the angle between them. The result is a number in [0,1] where the higher this value is, the more similarity between two vectors.

$$dC_{(A,B)} = \frac{\sum_{w \in A \cap B} P_A(w) \times P_B(w)}{\sqrt{\sum_{w \in A \cup B} P_A(w)^2} \times \sqrt{\sum_{w \in A \cup B} P_B(w)^2}}$$

It is one of the most commonly used method in information retrieval to determine how similar two documents are (often using $tf - idf$ instead of absolute/relative frequencies).

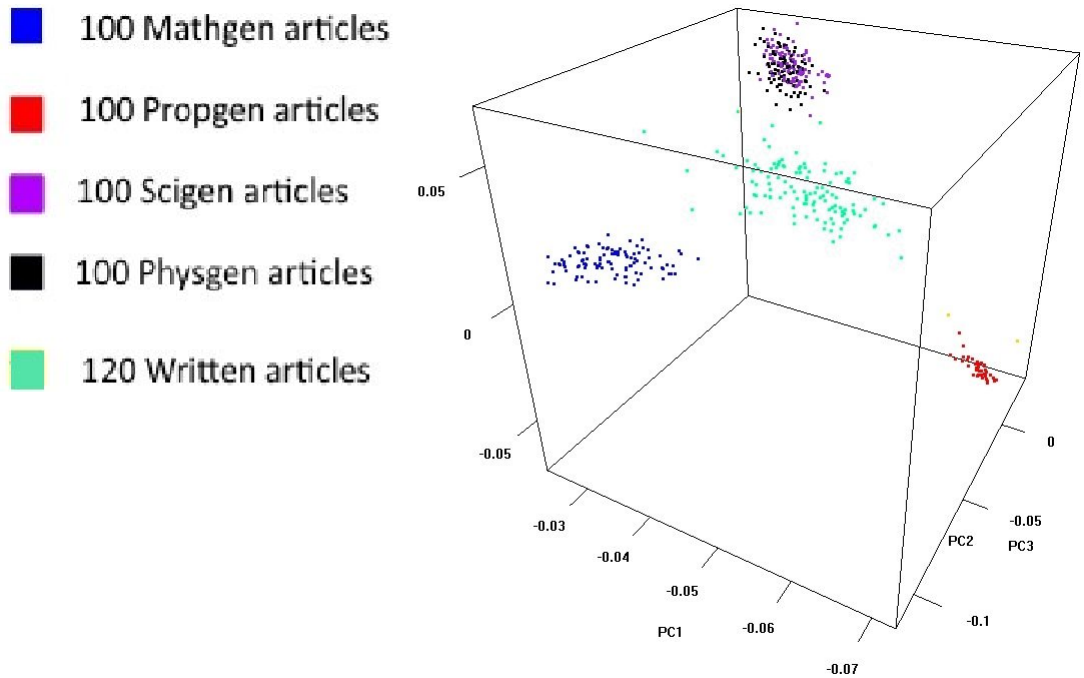


Figure 3.3: Principal-component analysis plots of stop-words in different corpora as a reproduction of the figure in [Ginsparg, 2014]

Textual Distance [Labbé and Labbé, 2013]: as presented in previous sections, this is a method to compute the differences in the proportion of word tokens between two texts. The distance between two texts A and B where $N_A < N_B$ is:

$$d_{(A,B)} = \frac{\sum_{w \in A \cap B} |F_A(w) - \frac{N_A}{N_B} F_B(w)|}{2N_A} = \frac{1}{2} \sum_{w \in A \cap B} \left| \frac{F_A(w)}{N_A} - \frac{F_B(w)}{N_B} \right|$$

where $d_{(A,B)} = 0$ means A and B share the same word distribution and $d_{(A,B)} = 1$ means there is no common word in A and B.

This method is particularly useful for multiple tasks such as authorship attribution [Labbé and Labbé, 2006], and classifying text from different fields or genes.

Nearest Neighbor Classification:[Altman, 1992] These methods to calculate similarity would work quite well in tandem with nearest neighbor classification. In this method of classification, corpora of different types of automatically generated papers were used as sample corpora. Then for each paper under test, the distance/similarity to each and every other paper in the sample corpora is calculated, the minimum value of these distance/similarity is denoted as the nearest distance/similarity to their nearest neighbor inside the sample corpora.

Specifically, for each generator (SCIgen, Physgen, Mathgen and Propgen) a set of 400 texts were used as tuning corpora $\mathscr{T}$ (a total of 1600 texts). A test corpus $\mathscr{C}$ which is composed of an

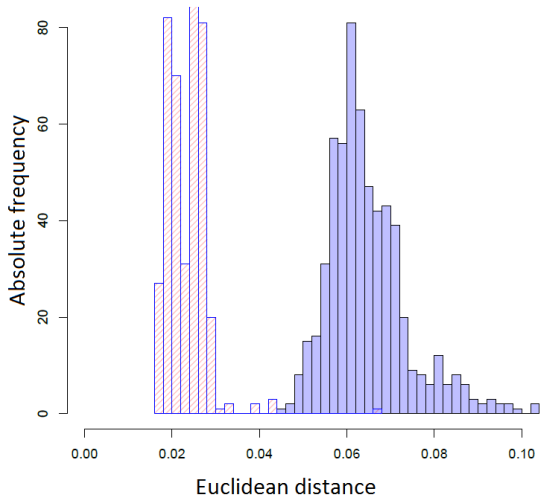


Figure 3.4: Distribution of Euclidean distance to the nearest neighbor of generated text (red) and genuine text (blue) (from [Nguyen and Labbé, 2016])

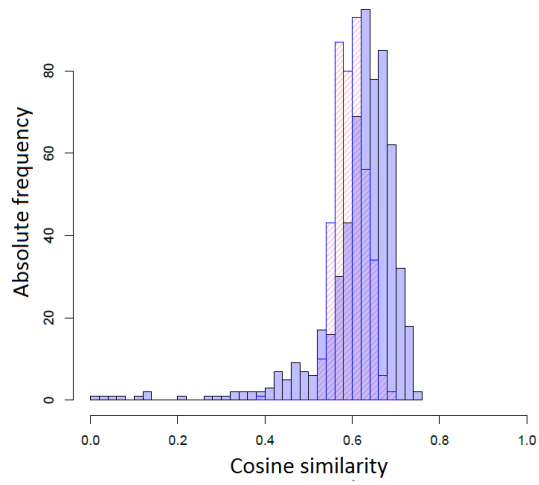


Figure 3.5: Distribution of cosine similarity to the nearest neighbor of generated texts (red) and genuine texts (blue) (from [Nguyen and Labbé, 2016])

extra 100 texts per generator (400 automatically generated texts) and 100 genuinely written papers taken randomly from ArXiv. Then for each document A in $\mathscr{C}$, all the distance/similarity δ to other documents in $\mathscr{T}$ is calculated. The minimal value of these distance/similarity is considered as the Min distance to the tuning corpus. Or, for each $A \in \mathscr{C}$ then its nearest neighbor is $B\ in \mathscr{T}$ such as:

$$Min_{\delta}(A, \mathscr{T}) = Min_{B \in \mathscr{T}}(\delta_{(A,B)})$$

and

$$Max_{\delta}(A, \mathscr{T}) = Max_{B \in \mathscr{T}}(\delta_{(A,B)})$$

Where the *distance* can be defined as any of the distance/similarity calculation. The distribution of these calculations for minimal distance are shown in Figures 3.4, 3.5 and 3.6. And based on these Figures, textual distance along with closest neighbor classification have been adopted in SciDetect to find automatically generated documents, which will be presented in the next section.

3.3 SciDetect

In this section we present our SciDetect system, based on figures 3.4, 3.5 and 3.6. It can be seen that inter-textual distance, along with the nearest neighbor provides the best separation of generated vs genuinely written paper. Thus, this method is adopted in our system to classify automatically generated text.

Furthermore, by practice, we see that inter-textual distance is quite sensitive to the length of the document, so, to avoid mis-classifications caused by text length, texts shorter than 10,000 characters

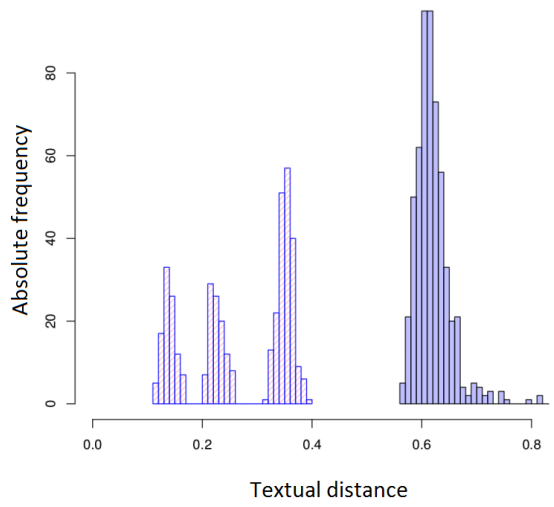


Figure 3.6: Distribution of textual distance to the nearest neighbor of generated texts (red) and genuine texts (blue) (from [Nguyen and Labbé, 2016])

were ignored and texts longer than 30,000 characters were split into smaller parts. To determine the genuineness of a text, we used different thresholds for each type of generator. Table 3.1 shows detailed statistical information about the observed distances in Figure 3.6.

Along with that, to determine an upper threshold for genuine texts, a set of 8,200 genuine papers from various fields were used. The nearest neighbor for each genuine text was computed using the same sample sets.

The first two rows of Table 3.1 show that, for a genuine paper, the minimal distance to the tuning corpus in the sample set (0.52) is always greater than the maximal distance to the tuning corpus of a fake paper (0.40). It is also easy to see that documents from different generators are quite well separated since the min distance to the tuning corpus of a generated document is always of the same type.

	Scigen	Physgen	Mathgen	Propgen	Genuine
$Min_\delta(A, \mathscr{T})$	0.30	0.31	0.19	0.11	**0.52**
$Max_\delta(A, \mathscr{T})$	**0.40**	0.39	0.28	0.22	0.99
$Mean_\delta(A, \mathscr{T})$	0.35	0.35	0.22	0.14	0.69
$Stdev\sigma_\delta(A, \mathscr{T})$	0.01	0.01	0.01	0.0	0.12
$Median_\delta(A, \mathscr{T})$	0.35	0.35	0.22	0.14	0.64

Table 3.1: Statistic summary of textual distances between papers and their nearest neighbor (the nearest neighbor is always of the same kind).

By observing the results, we concluded that there would always be a close grouping of the generated texts that are separated from the group of real texts with a considerable gap in between. It is safe to say that we can classify the text based on thresholds. Thus, two thresholds for each generator were set: a lower threshold for generated papers based on the second row of Table 3.1 and an upper threshold for genuine papers (varying from 0.52 to 0.56 depending on the generator).

Hence, a paper can be identified as possibly generated in two different ways. First, if the distance is lower than the specific threshold for a generated paper then it is considered as a confirmed case of a generated paper. Second, if the distance is between the thresholds for a generated and genuine paper, it is considered as a suspicious case.

3.4 Comparative Evaluation Between Different Methods

To thoroughly evaluate SciDetect and other methods, we decided to conduct a comparative test using different known methods

3.4.1 Test Candidates

In this section, various methods to detect an automatically generated document that is or has been working in a real-life environment are presented.

Pattern Matching: We were able to obtain the software from Springer as presented earlier and used it as a test candidate to detect generated documents.

Kullback-Leibler Divergence and Nearest Neighbor Classification: As presented before, this method seems to be currently used by ArXiv. We implemented our own system that uses a list Sw of 571 stop-words [Feinerer et al., 2008] to classify texts. A profile for the average distribution of the stop word frequencies for each generator was created using the same 400 generated texts in the sample corpora of SciDetect.

Two thresholds for each generator were also established in the same manner as in Section 4 namely, a generated threshold for the maximum KL-divergence between a profile and a generated text from the test corpus; and a written threshold with the minimum KL-divergence between a profile and genuinely written texts.

SciDetect: We would also like to verify the usefulness of our SciDetect system as presented in Section 3.3.

3.4.2 Test Corpora

We used three different corpora to conduct the test:

- Corpus $\mathscr{X}$: 100 texts from known generators (25 for each type of generator) without any modification.

- Corpus $\mathscr{Y}$: 100 generated texts (25 from each generator) that have been modified by randomly changing a word every two to nine words with a word taken from a genuine research paper. The aim of this corpus is to test the robustness of these methods against not only pure generated texts but also modified versions which have somewhat different word distribution compared to the samples.

- Corpus $\mathscr{Z}$: 100 real texts with a different length ranging from two pages to more than 100 pages.

method	corpus	True Positive		False Positive		True Negative	False Negative
		confirm	suspect	confirm	suspect		
Pattern Matching	$\mathscr{X}$	25%	4%	0	0	0	71%
	$\mathscr{Y}$	8%	16%	0	0	0	76%
	$\mathscr{Z}$	0	0	0	1%	99%	0
Kullback-Leibler Divergence	$\mathscr{X}$	87%	13%	0	0	0	0
	$\mathscr{Y}$	79%	21%	0	0	0	0
	$\mathscr{Z}$	0	0	0	4%	96%	0
SciDetect	$\mathscr{X}$	100%	0	0	0	0	0
	$\mathscr{Y}$	100%	0	0	0	0	0
	$\mathscr{Z}$	0	0	0	0	100%	0

Table 3.2: Results of the different methods on the three corpora

3.4.3 Results

These experiments aim at determining the performance of the different methods for detecting generated papers. The results are shown in Table 3.2 whereby:

- True positive is a correctly identified generated paper. A paper can be identified as positive in two different ways: if the distance/score is lower than a specific threshold for a generated paper, then it is considered as a confirmed case of positive; or if the distance/score is between the threshold for a generated and a genuine paper, it is considered as a suspected case.
- False positive is a genuine paper that has been identified as generated (or suspected).
- True negative is a correctly identified genuine paper.
- False negative is a generated paper that has been identified as genuine or as an error during processing.

Close study of these results highlights several interesting aspects. Considering the current state of generators, current classifiers all work relatively well (all achieved a perfect precision rate). Difficult cases (Corpus $\mathscr{Y}$) were marked as suspicious thus requiring further investigation. Particularly, SciDetect was proven as the most reliable method–all tests passed at 100%. Furthermore, even though pattern matching was designed to only match SCIgen patterns, it was able to recognize three papers from Scigen-Physics as suspected SCIgen; however, when applied to Corpus $\mathscr{Y}$, one modified SCIgen paper was mistakenly listed as genuine. One case of a false positive in the pattern checker with Corpus $\mathscr{Z}$ was caused by a large file with more than 110 pages, which triggered an out of memory error.

3.5 Detecting Markov and RNN Text with SciDetect and SciDetect Robustness

SciDetect and other previously presented methods are all focused on detecting a fully generated text by some kind of PCFG with some pre-determined sentence structures. However, Section 2 has shown that there is more than one way to reproduce text. Thus, this section shows some interesting results of detection for texts that were automatically generated using a RNN or a Markov model using SciDetect.

3.5.1 Detecting Markov and RNN Text with SciDetect

Currently, we are not aware of any attempt to detect automatically generated text using RNN or Markov model. However, it is expected that the generated texts should have some common statistical distribution with the original source that they were learned on. To verify this assumption, SciDetect was used to confirm the possibility of classifying automatically generated document given a known source.

Again, a Markov model with depth three and a recurrent neural network on character level are used to generate 100 documents each. Markov model generated texts have randomized lengths from five to ten thousand word tokens, while RNN texts have 50 to 250 thousand characters. Each is also generated with a random "seed text" which is a sentence taken from scientific papers. Then half of these generated texts are used as a sample corpus to detect the other half in a mixed corpus that includes generated text, genuinely written texts and the original texts that the Markov model or RNN were learned on. More specifically, the same experiment as in Section 3.2 is repeated. In this test several corpora were used.

- Markov tuning and RNN tuning: Each contains 50 automatically generated documents from the corresponding method.
- Markov test and RNN test: Each is composed of the same 100 genuinely written documents, the original documents that were used to learn the distribution and another 50 automatically generated documents from each method.

In Figure 3.7, the distribution of minimal distance from each document in the Markov test corpus to the Markov sample corpus is shown and the same for Figure 3.8, but with the RNN test and RNN sample.

Figure 3.7 and 3.8 show that generated text from a same source, even with different seed text, all end up having quite similar textual distance (or word distribution) compared to genuinely written texts. Furthermore, with the Markov model, it is also possible to see that the original source of the model is closer to the generated ones compared to genuinely written texts. From the graph, it is possible to set simple thresholds to classify a document as automatically generated by a Markov model or a RNN, particularly in our case a threshold is set at 0.53 for the former and 0.55 for the later. Using these thresholds, the results in Table 3.3 can be obtained.

Considering that **precision** or **recall** can be easily manipulated by using different size corpora. For example, if a test corpus is composed of ten thousand automatically generated texts and only a thousand genuinely written ones. And if a classification method just marked everything as automatically generated, then the Precision, as well as Recall, are both very high, even though all the genuinely written documents are marked as automatically generated.

However, a false positive rate, which is the probability of a genuinely written text marked as generated, and vice versa for false negative rate are independent from the corpus size, so they are used to fairly represent the results.

This once again confirms that if somehow the source or a sample set of generated documents from the same source can be obtained, it is possible to detect generated documents in a mixed corpus of different text types.

3.5.2 Testing SciDetect's Robustness

To test the robustness of the family of methods based on words statistics, the SciDetect system [Nguyen and Labbé, 2016] is used in a less than ideal condition where only part of a document

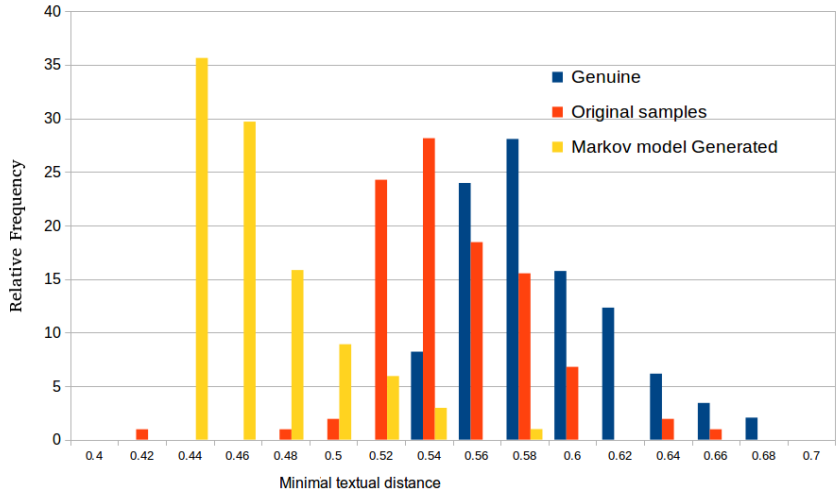


Figure 3.7: Relative frequency of textual distance for different test type using Markov model sample

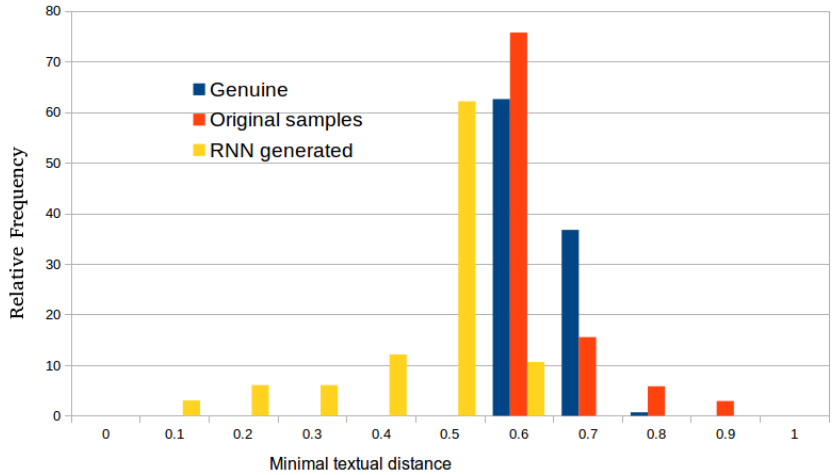


Figure 3.8: Relative frequency of textual distance for different test type using RNN

is generated by a known generator. Ten genuinely written texts, each with about 5000 word tokens are used, and then we gradually replace 250 words at a time with a same size chunk of text from other SCIgen papers to discover the limitation of the system. Then this corpus $\mathscr{A}$ of modified texts is run through SciDetect to obtain the average minimal distance δ from $\mathscr{A}$ to the samples corpus $\mathscr{T}$ of SciDetect.

SciDetect uses thresholds to determine if a text is automatically generated or not. Specifically, if the minimal distance of a document to the sample corpus is higher than 0.56, then it is considered as genuine. If the distance is between 0.56 and 0.45, then it is considered as a suspicious case of automatically generated, and lower than 0.45 is a confirmed case of automatically generated.

	False positive rate	False negative rate	Precision	Recall
Markov model	0.28	0.04	0.77	0.96
RNN	0.0	0.08	1	0.92

Table 3.3: False positive, false negative rate as well as precision and recall for SciDetect with Markov model and RNN.

SciDetect result / percentage of genuine text in $\mathscr{A}$	average $Min_\delta(\mathscr{A}\mathscr{T})$	Percentage of detected texts
100%	0.72	0%
95%	0.70	0%
90%	0.67	0%
85%	0.65	0%
80%	0.63	0%
75%	0.59	10%
70%	0.56	30%
65%	0.55	50%
60%	0.52	90%
55%	0.48	100%
50%	0.45	100%

Table 3.4: Robustness of SciDetect

Table 3.4 shows that SciDetect can only detect a partially generated document if the automatically generated part reaches a certain length. Particularly in our test, normally, as soon as about one third of the document is automatically generated, SciDetect would be able to mark half of the partially generated texts as suspected cases of automatically generated, and if about half or more of the document is automatically generated, then it is possible to positively classify all the partly generated texts as being generated.

Considering the case of the paper by Navin Kabra (Figure A.1 in the Appendix) that was mentioned in Section 1, this paper is about two thirds automatically generated but it is still very short (only 2000 words). SciDetect [Nguyen and Labbé, 2016] classifies this paper as a suspicious case of automatically generated (distance to the samples corpus $\mathscr{T}$ is 0.49). Other cases of known partially generated paper in Figure 1.1 and in the Appendix (Figure A.2 in the Appendix) are also classified as suspicious by SciDetect (0.49 and 0.46 respectively).

This is understandable since SciDetect is based on word distribution, thus without a substantial amount of generated text, the distribution would not have any significant impact. This would further open a different problem when one needs to confidently detect a particularly generated document where only a small part or even a few sentences are automatically generated. This problem will be tackled within the next Chapter.

3.6 Summary

This chapter has shown that there are multiple methods to discover automatically generated scientific papers. They range from simply checking the name of the references to building a complex network based on word connections and each has a different level of success.

However, we presented our SciDetect system to automatically classify an automatically generated paper which is based on inter-textual distance along with nearest neighbor classification and it has been proven to be able to provide a reliable result. Furthermore, it has also been proven that if some samples can be obtained, it is possible to detect texts that were generated using a Markov model or a recurrence neural network.

SciDetect is freely available online as an open source Java program. It can be obtained from a Git repository[2] along with fully implemented documentation (as in the Appendices) as well as Javadoc for each class.

Nonetheless, facing a partially generated paper where only a small fraction or even only a small section of the paper is automatically generated, current methods are impractical. Thus Chapter 4 will present a method to tackle partially automatically generated papers based on sentence structure.

[2] http://scidetect.forge.imag.fr/

Chapter 4

Detecting a Partially Automatically Generated Document

Chapter 3 has shown that it is possible to automatically detect fully generated documents. However, it was also shown that current methods are ineffective when facing a document which was only partly generated or contains only a small part of automatically generated text.

This problem may be seen as hypothetical but we believe that real life examples show its actual nature. Figure 1.1 shows an example of a partially generated document that was accepted and sold by IEEE as being a peer-reviewed genuine document. Another example is when Navin Kabra[1] got one of his papers (Figure A.1) published in an *international conference* called "International Conference on Recent Innovations in Engineering, Science & Technology" even though it was mostly automatically generated by SCIgen, and at the same time contained dialogue from a movie. Another example of this kind of paper is given in Appendix (Figure A.2). This latter being a partially generated paper that was submitted (probably as a prank) and not accepted to a Springer conference (ICAART 2014). In a context where publishers are struggling in trusting editors, nobody can confidently say that there are no other cases where only a small proportion of generated text was sneaked into a genuinely written document as a placeholder for an unfinished section, to extend the document to an appropriate length, or just to "game the system."

Current detection methods are still somewhat limited since they are not able to detect a small quantity of automatically generated text within a large body of genuinely written text (e.g., only several sentences or a paragraph out of a whole genuinely written paper). We believe that this is the first attempt to develop a system to combat against such problems.

We investigated a classification approach aimed at characterizing the main features of generated sentences, so they can be flagged individually. Current automatic paper generators make use of sets of rules to generate sentences. Sentences generated using a particular rule might have a similar grammatical form and differentiate only in the words chosen at random. Where each sentence can be separated into Verb Phrase (VP), Noun Phrase (NP). Then the phrases can be separated again

[1] https://smritiweb.com/navin/education-2/how-i-published-a-fake-paper-and-why-it-is-the-fault-of-our-education-system

and again at deeper level as Noun, Verb, Adjective, etc.... This might make one think that sentences with a similar structure and/or word pairs might be related to each other. Thus, we investigate an approach that measures the similarity between sentences based on their grammatical structure (parse tree) without paying too much attention to the words used in the sentences

In particular, Section 4.1 gives a brief explanation of a parse tree and how the grammatical structure tree can be used to calculate the similarity between sentences. Then Section 4.2 describes our definition of grammatical structure similarity and how our system to detect automatically generated sentences was built. Section 4.4 details the validation of our system in comparison with other approaches, namely a pattern checker and some traditional machine learning techniques. The chapter ends with Section 4.5 where a short conclusion and some limitations are stated.

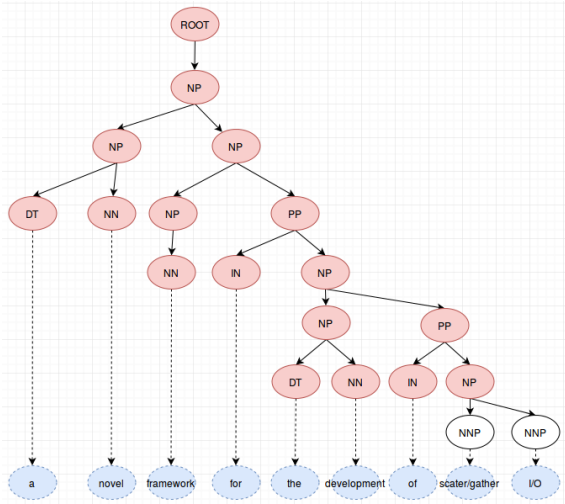
4.1 Using Parse Tree on Sentence Similarity

Dependency tree or parse tree is a tree that represents the syntactic structure of a sentence or a phrase (Example 7 and 8).

Over the years, there have been multiple proposals using different types of parse trees to discover the similarity between sentences and from there to apply the similarity measurement for different goals.

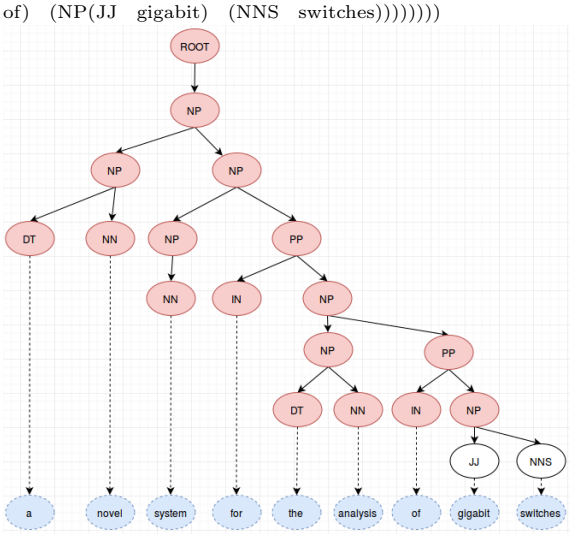
Example 7 A parse tree in different forms for the phrase: "a novel framework for the development of scatter/gather I/O".

(ROOT(NP(NP(DT a) (NN novel)) (NP(NP(NN system)) (PP(IN for) (NP(NP(DT the) (NN analysis)) (PP(IN of) (NP(NNP scater/gather) (NNP I/O))))))))



Example 8 A parse tree in different forms for the phrase: "a novel system for the analysis of gigabit switches".

(ROOT(NP(NP(DT a) (NN novel)) (NP(NP(NN system)) (PP(IN for) (NP(NP(DT the) (NN analysis)) (PP(IN of) (NP(JJ gigabit) (NNS switches))))))))



.

4.1.1 Using Syntactic Tree to Discover Plagiarism

[Zubarev and Sochenkov, 2014] uses their sentence similarity measurement in Exactus [Sochenkov et al., 2016] to discover plagiarism. This measurement uses different characteristics from the sentence including TF-IDF, IDF overlap and a syntactic similarity measurement where the syntactic links between pairs of words from different sentences are measured. Specifically, for two sentences s_e and s_t from two documents d_e and d_t, the similarity is defined as a combination of three similarity I_1, I_2 and I_3 as follows:

IDF overlap measure: For a set of documents D and $m(w_e, D)$ is the number of documents that contain the word w_e and $N_{(s_e,s_t)}$ is a set of word pairs with the same lemma from two sentences s_e and s_t.

$$I_1(s_e, s_t) = \sum_{(w_e,w_t)\in N_{(s_e,s_t)}} IDF_{(w_e)}$$
$$IDF(w_e) = log_{|D|}(\frac{|D|}{m(w_e,D)})$$

TF-IDF-based similarity measure: For a document d_t with $|d_t|$ words, the number of times a word w_t appears in d_t is $k(w_t, d_t)$, and a function $f(w_e, w_t)$ for penalty for a mismatch of w_e, w_t forms (same lexeme but different form (e.g.,: play, played, plays, playing, etc.):

$$f(w_e, w_t) = \begin{cases} 1, & \text{if } w_e, w_t \text{ have a same form} \\ 0.8, & \text{otherwise} \end{cases}$$

then the TF-IDF-based measure is defined as

$$I_2(s_e, s_t) = \sum_{(w_e,w_t)\in N_{(s_e,s_t)}} f(w_e, w_t) IDF(w_e) TF(w_t, d_t)$$
$$TF(w_t, d_t) = log_{|d_t|}(k(w_t, d_t))$$

Sentence syntactic similarity measure: The sentence similarity is defined based on the syntactic dependency tree from each sentence. $Syn(s_e)$ is defined as a set of triplets (w_h, σ, w_d) where w_h, w_d are head and dependent word, and σ is the type of syntactic relation. Then the syntactic similarity is:

$$I_3(s_e, s_t) = \frac{\sum_{(w_h,\sigma,w_d)\in(Syn(S_e)\cap Syn(S_t))} IDF(w_h)}{\sum_{(w_h,\sigma,w_d)\in Syn(S_e)} IDF(w_h)}$$

In simpler terms this means that it is the sum of IDF of the head word from all common triplets between sentence s_e and sentence s_t over the sum of IDF of all the head words in sentence s_e.

Overall sentence similarity: From all those previously defined measurements, the overall similarity between sentences is defined as:

$$Sim(s_e, s_t) = WIdf \times I_1(s_e, s_t) + WTfIdf \times I_2(s_e, s_t) + Wsynt \times I_3(s_e, s_t)$$

Where $WIdf, WTfIdf, WSynt$ are custom weights for distribution of each type of similarity and needs to be manually tuned for different corpus. From this overall sentence similarity, a threshold can be set to determine if a sentence is plagiarized or not. Using this method, the authors were able to obtain the second highest score for the plagiarism detection track at the PAN workshop in 2014[2].

This method seems to perform very well with plagiarism detection where sentences are quite similar on a lemma level. However, in our specific use case for PCFG generated sentences where words are chosen at random, then the approach is impractical.

[2] http://pan.webis.de/clef14/pan14-web/

4.1.2 Relation Extraction Based on Common Tree Segments

[Culotta and Sorensen, 2004] proposes a method to estimate the similarity between sentences using the number of common tree segments in their augmented parse tree. This augmented parse tree is first created by MXPOST [3], a maximum entropy parser. Then for each pair of entities in a sentence, a smallest common sub-tree in the parse tree is found, and then it is used to create the augmented parse tree where the nodes are represented as a feature vector instead of an entity in the sentence.

The similarity score between two augmented trees is then defined using a tree kernel function $K(T_1, T_2)$. This function is broken down into two smaller functions that represent the matching $m(t_i, t_j)$ and the similarity $s(t_i, t_j)$ between two sentences t_i and t_j with their feature vectors noted $\phi(t_i)$ and $\phi(t_j)$ respectively. Then t_i and t_j would be a match if they share a same feature vector or:

$$m(t_i, t_j) = \begin{cases} 1, & \text{if } \phi(t_i) = \phi(t_j) \\ 0, & \text{otherwise} \end{cases}$$

and the similarity between t_i and t_j would be:

$$s(t_i, t_j) = \sum_{v_q \in \phi(t_i)} \sum_{v_r \in \phi(t_j)} C(v_q, v_r)$$

Where $C(v_q, v_r)$ is some compatibility function between two features values such as:

$$C(v_q, v_r) = \begin{cases} 1, & \text{if } v_q = v_r \\ 0, & \text{otherwise} \end{cases}$$

The final kernel function for the similarity between two trees T_1, T_2 with root r_1, r_2 respectively is:

$$K(T_1, T_2) = \begin{cases} 0, & \text{if } m(r_1, r_2) = 0 \\ s(r_1, r_2) + K_c(r_1[c], r_2[c]), & \text{otherwise} \end{cases}$$

This means the function would start from the root of the trees; if they are a match, then the similarity between them and any other matching child is calculated using a recurrent loop. Using this method along with SVM on the Automatic Content Exaction corpus by the National Institute for Standards and Technology, the authors were able to show that a dependency tree kernel has a big improvement when compared to just a bag of word kernels in a relation extraction task.

However, again as in the previous method, the comparison would start with a pair of similar roots, which is typically not the case for PCFG generated sentences, thus simply applying it to our needs might not work.

4.1.3 Textual Entailment with Parse Tree DLSITE-2

Textual entailment is the process to determine if a hypothesis can be deducted from another text.The DLSITE-2 system [Wang and Neumann, 2007] aims at determining textual entailment using a syntactic parse tree.

In this work, sentences are selected based on words with significant grammatical value likes nouns, verbs, adjectives, etc. Sentences with the same or similar significant grammatically value

[3] ftp://ftp.cis.upenn.edu/pub/adwait/jmx/

Tag	Tag
ADJP	ADP
NP	PP
S	SBAR, SBARQ
LS	VB, VBD, VBG, VBN, VBP, VBZ
ETC...	ETC...

Table 4.1: Some examples of tag groups with similar grammatical category

words are parsed to syntactic trees. These trees are then filtered to reduce irrelevant data and processing time; only verbs, nouns, numbers, adjectives, adverbs and noun-noun modifiers are kept. They are then compared to detect if one is contained by another. A tree T_1 is considered to be contained by another tree T_2 if, and only if, all nodes and branches of T_1 is present in T_2. This process is started at the roots, and if they are a match, then it is extended to their respective child nodes. The matching function does not require the tokens to be the same but can be more flexible by using a word similarity measure such as WordNet::Similarity tool [Pedersen et al., 2004] with a certain threshold.

Using this proposal on a set of text-hypothesis pairs from the Second PASCAL Recognizing Textual Entailment Challenge (RTE-2) [Herrera et al., 2006], the authors were able to achieve an accuracy of 60.75 which was about 10% better than the base line.

This approach is heavily dependent on the vocabulary of the sentences under test. In the case of PCFG even sentences generated with the same rule are not guaranteed to share the same vocabulary, thus it might not be best suited.

4.1.4 Sentence similarity with Parse Tree

Parse tree along with common words are also used by [Durán et al., 2014] to determine the similarity between sentences. They propose a method to search for semantic relation based on the exploding of Parse tree starting from pairs of similar words. For each sentence, the syntactic dependency tree is created using Stanford Parser [Klein and Manning, 2003], and two trees will be compared if they share at least two pairs of common words. From a pair of common words, their respective ancestors from each tree are compared and if they are similar (Table 4.1). If so, the process is continued until the root is reached.

From other common word pairs, the same process is repeated and if a connection is formed, it will be used as a common sub-tree. Those common sub-trees are then used to calculate a final similarity between two sentences with different weights for different types of nodes.

For each common subtree C with r nodes of tree A with p nodes and tree B with q nodes, the degree of similarity is calculated using:

$$S_t = \frac{\sum_{k=1}^{r} W_{nCk}(\sum_{i=1}^{p} WnAi + \sum_{j=1}^{q} WnBi)}{2\sum_{i=1}^{p} WnAi \sum_{j=1}^{q} WnBj}$$

Where W_n is the weight for the type of node that can be found in Table 4.2.

If two sentences share more than one common sub-tree then the final similarity between them is calculated as the average of all the degrees of similarity for those common sub-trees.

In general, this method would be much better fitted to the aim of this chapter compared to other previous approaches. However, since similar words are heavily weighted compared to other

Node type	Weight
Words	4
ADJP, ADVP, NP, PP, VP	3
WHADVP, WHNP, WHPP	3
NN, NNS, NNP, NPS	2
VB, VBD, VBG, VBN, VBP, VBZ	2
Others	1

Table 4.2: Some examples of tag groups with similar grammatical category

node types, that might also make some automatically generated sentences different from each other. Thus, the next section will present our definition of grammatical structure similarity and how it is used to compare sentences.

4.2 Definition of Grammatical Structure Similarity and Building of the System

All previously mentioned methods to calculate similarity performed quite well in their respective sub-domain. However, they might not fit our need to detect short segments of PCFG generated text since they are either too focused on pairs of common words in the sentences or are computationally expensive when every sentence needs to be parsed.

So, this section first presents our proposal for calculating the similarity between sentences as a means to detect sentences that only depend on the structure of the sentences instead of pairs of similar words in 4.2.1. Then, some corpora are presented in 4.2.2 and Section 4.2.3 extends the definition of Grammatical Structure Similarity to be used with the previously presented corpora to demonstrate the effectiveness of different sized corpus, also pointing out some typical mistakes by the system. Section 4.2.4 tries to limit the number of mistakes and also to reduce the amount of work needed by employing a Jaccard filter before processing.

4.2.1 Grammatical Structure Similarity

This section shows how data is handled and the definition of grammatical structure similarity.

For PDF documents in the test corpus, they are first converted to plain text, then normalized (decapitalize, remove numbers, symbols, non-conventional characters, etc.). Later these texts are separated into sentences, and each sentence is parsed using Stanford Parser [Klein and Manning, 2003] to obtain a parse tree. Since in our case, the keywords have little to no value in deciding the similarity between sentences, they are removed from the structure, and only the nodes are kept (Example 9). These parse trees are then compared to the PCFG corpus of known parse trees from pre-processing generated sentences using a recursive loop to find the biggest possible sub-tree match of the tree structure.

Example 9 The parse tree in example 7 would be considered only as:

(ROOT(NP(NP(DT) (NN)) (NP(NP(NN)) (PP(IN) (NP(NP(DT) (NN)) (PP (IN) (NP(NNP) (NNP))))))))

Once a similar structure is found, the similarity between them needs to be quantified. So, the **grammatical structure similarity** is defined as follows:

Corpus size	nb of sentences	nb of distinct sentences	nb of distinct parse trees
80 documents	12.1k	9.2k	8k
160 documents	25.5k	18.2k	14.8k
320 documents	45.5k	33.2k	26.9k

Table 4.3: Number of sentences, distinct sentences and parse trees for different $\mathscr{T}$ corpus size

Definition 1. Grammatical structure similarity (GSS):

Let N_A be the number of nodes in the parse tree T_A of sentence A, N_B be the number of nodes in the parse tree T_B of sentence B, and N_{AB} be the number of nodes in the biggest common sub-tree of T_A and T_B. Then the Grammatical Structure Similarity between A and B is defined as:

$$GSS_{(A,B)} = \frac{2*N_{AB}}{N_A+N_B} \in [0,1]$$

Example 10 **Grammatical Structure Similarity** between Example 7 and Example 8

$$GSS_{(E_{21}/E_{22})} = \frac{2*17}{19+19} = 0.89$$

This means that if the GSS between two sentences is 1, they share the same sentence structure (might be different on the lemma level), if the $GSS = 0$, it means they do not share any common sub-tree or even node's type.

In our proposal, the computation is quite expensive since each sentence needs to go through the parser and is compared with all samples in the PCFG corpus. This will be explored further in the following section.

4.2.2 Corpora

Multiple text corpora are used for later testing purposes and this section will give a detailed description about them.

PCFG Corpus $\mathscr{T}$: PCFG corpora of different sizes were used as samples and tried to learn the different parse tree structures from the generators. Table 4.3 shows the correlation between the size of the corpus (evenly distributed between four generators) with the number of sentences and the number of distinct parse trees that can be obtained from those sentences.

Even though the number of distinct sentences and trees increased steadily, it is possible that there are only small variations between them and this will be tested in Section 4.2.3 later.

Test Corpus $\mathscr{C}$: This corpus is composed of 4 smaller corpora each containing 100 texts from an automatic generator and a real corpus of 100 genuinely human written texts which were selected at random from different fields. These smaller corpora are noted as $\mathscr{C}$.real, $\mathscr{C}$.Scigen, $\mathscr{C}$.physgen, $\mathscr{C}$.mathgen and $\mathscr{C}$.propgen. This resulted in about 110k sentences used as the test corpus $\mathscr{C}$.

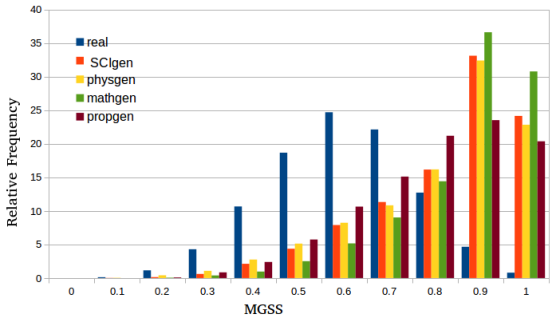
4.2.3 Effectiveness of GSS for Different PCFG Corpora

Our hypothesis is that even though the number of distinct sentences as well as parse trees seem quite numerous (Table 4.3), most of them should also be somewhat similar to each other. Only a small proportion of the sentences are different. To verify this, the maximum GSS (MGSS) of all sentences in the test corpus for three different size PCFG corpora $\mathscr{T}$ (that were computed and presented earlier), and the results are shown in Figure 4.1.

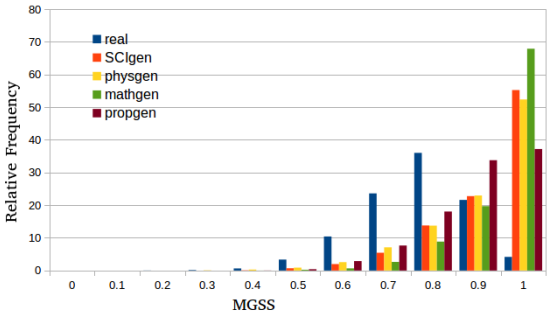
Definition 2. Maximum Grammatical Structure Similarity (MGSS): For a sentence A in the test corpus ($\mathscr{C}$), the MGSS between A and the PCFG corpus ($\mathscr{T}$) is:

$$MGSS_{(A,\mathscr{T})} = Max_{(B \in \mathscr{T})}(GSS_{A,B})$$

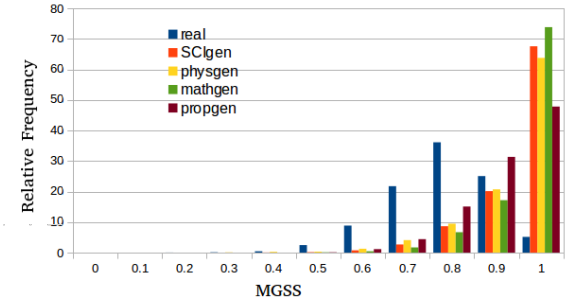
When comparing the PCFG corpus $\mathscr{T}$ of size 80 with the others, with more samples in the PCFG corpus $\mathscr{T}$, it is possible to find a higher GSS match for generated sentences. However, comparing the $\mathscr{T}$ of size 160 and 320, it is difficult to see any significant difference. This suggests that the previous hypothesis is true. Even though the size of the $\mathscr{T}$ corpus was doubled, it did not double the match rate because most of the additional parse tree structures have very little differences with what has already been obtained in the smaller size corpus. Subsequently, from now on, only the PCFG corpus $\mathscr{T}$ of size 160 is used.



PCFG corpus $\mathscr{T}$ with 80 samples.



PCFG corpus $\mathscr{T}$ with 160 samples.



PCFG corpus $\mathscr{T}$ with 320 samples.

Figure 4.1: The distribution of MGSS for sentences in the test corpus $\mathscr{C}$ with the PCFG corpora $\mathscr{T}$

Figure 4.1 also shows that for SCIgen, physgen and mathgen, it is possible to find more than 50% of high matches (GSS higher than 0.9) as compared to less than 2% for genuinely written papers. Even though there is no clear separation for the score, it is easy to see that there are different distributions for genuinely written and generated ones. The curves for generated sentences lean very heavily toward the end of the histograms, thus making them stand out.

Furthermore, Table 4.4 shows some examples of genuinely written sentence with high GSS to other sentences in the PCFG corpus. Most of them are just common sentences that also appear in the PCFG corpus $\mathscr{T}$. To deal with such problems, the context of the sentence is considered, and this will be presented in Section 4.3 .

Genuinely written sentence	Sentence in PCFG corpus	Jaccard similarity	GSS
our main contributions are as follows	our main contributions are as follows	1	1
it is easy to see that	it is easy to see that	1	1
the states of this network are	the contributions of this work are as follows	0.4	0.89
the rest of the paper is organized as follows	the rest of this paper is organized as follows	0.88	1
the remainder of the paper is organized as follows	the rest of this paper is organized as follows	0.8	1
the proof of the claim can be found in appendix	useful survey of the subject can be found in	0.58	0.9
the interpretation of the walk is as follows	the rest of the paper proceeds as follows	0.45	1

Table 4.4: Some genuinely written sentences with high GSS score to a generated sentences

4.2.4 Sentence Filter Using Jaccard similarity

As mentioned before, the cost for parsing is quite expensive, and this raises the need to implement a filter to reduce the number of sentences that need to be parsed. To do such tasks, the Jaccard similarity is used (the number of common words over the total number of distinct words in two sentences). Figure 4.2 shows the Jaccard similarity between sentences in the test corpus with the sentences in the PCFG corpus that have the highest GSS to them.

Definition 3. Maximum Jaccard Similarity MJS: For a sentence A in the test corpus ($\mathscr{C}$), the Maximum Jaccard Similarity MJS between A and the PCFG corpus ($\mathscr{T}$) is:

$$MJS_{(A,\mathscr{T})} = Max_{(B \in \mathscr{T})}(Jaccard_{A,B})$$

As shown in Figure 4.2, the majority (more than 90%) of genuinely written sentences has MJS less than 0.3 to sentences in the PCFG corpus, while it was only about 20% for other types of generators (except about 40% for propgen). Subsequently, this would make 0.3 a good candidate for a threshold to be used in the filter since it is possible to keep a large number of "suspected generated" sentences while greatly reducing the number of "irrelevant" sentences.

Even though Jaccard similarity can filter out around 90% of the sentences, 20% of genuinely written sentences were also marked at the same time, and this would result in a large number of false positives. However, this filter significantly reduces the computational cost since it is no longer required to parse and compare the structure of each sentence with the whole PCFG corpus, only those that are similar to a generated sentence.

4.3 Fully Developed GSS System

Since the aim is to detect a small portion of automatically generated text, each sentence is considered along with its context, which includes the direct previous and next sentence to balance out special

cases. Thus, for each sentence in the test that is longer than 5 words and less than 35 words, the PCFG corpus $\mathscr{T}$ is used to find other sentences that have a Jaccard similarity higher than 0.3. Then the GSS between them is calculated to obtain the $MGSS$; the same process is repeated for the previous and the next sentences. The final GSS with context for the sentence is the average GSS of itself along with its direct neighbors.

The result for the Jaccard filter is shown in Table 4.5. It can be seen that the filter serves its purpose. Even though on average the number of sentences in a genuinely written paper is higher than in generated ones, only 20% of them go past the filter and need to be parsed as compared to 70% to more than 90% of sentences in automatically generated papers. This greatly reduced the processing time required, since, in reality, one would assume that an overwhelming number of sentences are genuinely written.

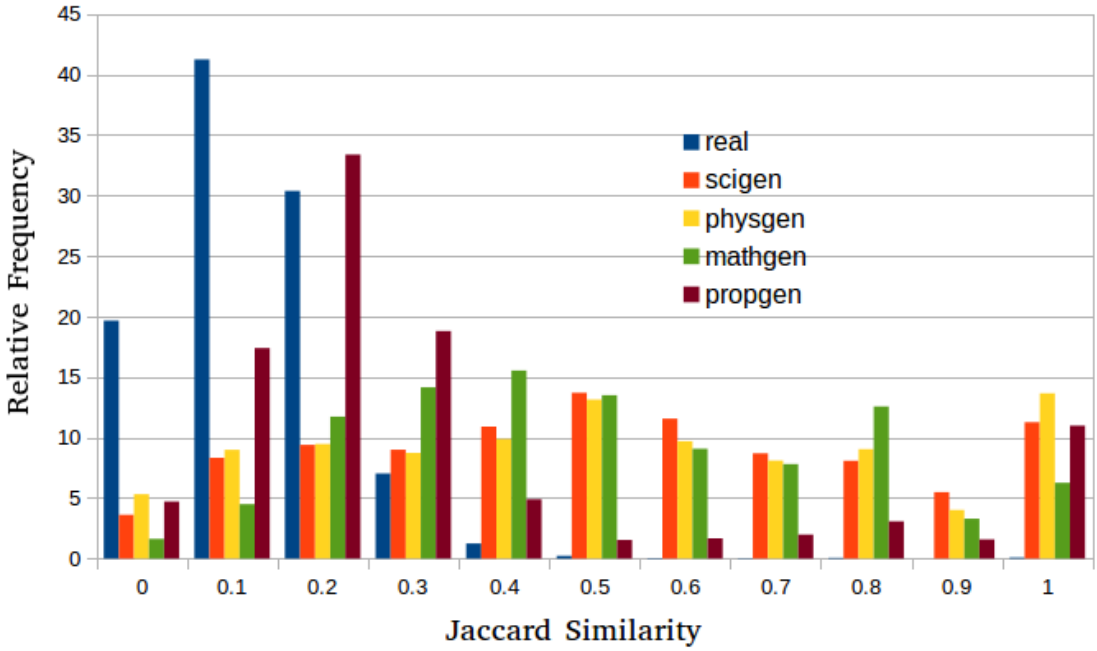


Figure 4.2: Relative frequency of maximum Jaccard similarity between different type of sentences to the PCFG corpus $\mathscr{T}$.

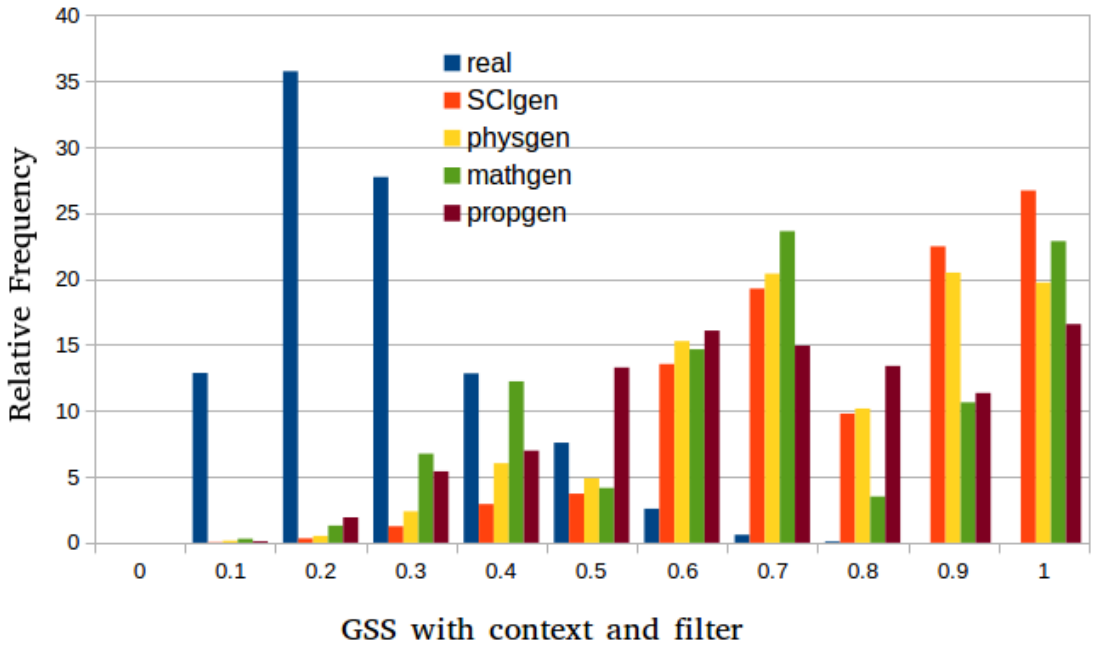


Figure 4.3: Relative frequency of $MGSS$ with context and Jaccard filter.

	Genuine	SCIgen	Physgen	Mathgen	Propgen
Avg number of sentences in a paper	192.4	87.2	81.7	174.0	88.9
Avg number of sentences that need to be parsed	38.8	80.0	73.8	161.4	62.9
Avg percentage of sentences that need to be parsed	20.2%	91.7%	90.2%	92.7%	70.7%

Table 4.5: Average number and Average percentage of sentences in a paper that need to be parsed

The result of the GSS system using the Jaccard filter and MGSS with context is shown in Figure 4.3. It shows that a majority (96.7%) of genuinely written sentences which pass through the filter have less than 0.5 GSS. On the other hand, for automatically generated sets of sentences if a threshold is set at 0.5, it is possible to detect more than 90% for SCIgen, physgen and about 75% mathgen and propgen.

4.3.1 Complexity of the System and Average Processing Time

There are several places that can be evaluated for complexity such as:

- Compare each sentence with all other sentences in the PCFG Corpus $\mathscr{T}$ using Jaccard similarity as a filter. The complexity of this step can be considered as $O(n)$ where n is the number of sentences in $\mathscr{T}$.

- If the sentence gets past the filter, it need to be parsed by the Stanford Parser. This is the

	False positive rate	False negative rate
SCIGen	0.0	0.68
SCIGen-Physic	0.0	0.86
Real	0.12	0.0

Table 4.6: False positive and false negative rate for Pattern checker method

costliest step with complexity $O(n^3)$ [Cer et al., 2010] where n is the number of words in the sentence.

– For each pair of sentences that go through the filter together, their parsed trees are compared to find the biggest common sub-tree. This cost $O(n^2)$ where n is the number of words in the sentence.

Eventually, the overall complexity of the system is $O(n^3)$. In practice, the run time for each document in our system [4] depends on the type of document. However, typically a fully genuinely written document would pass through every 10 to 20 seconds, while a fully generated one would take much longer from 60 to more than 90 seconds. This will not affect a real-life workflow much since most documents that need to be scanned are presumably all genuinely written.

Nevertheless, it is still necessary to thoroughly evaluate the performance of the system against some other techniques. This will be addressed in the next section.

4.4 Comparison with Other Methods

This section aims at comparing our newly developed system with some other well-known methods of detection or classification. This includes a simple pattern checker based on string token and different machine learning techniques.

4.4.1 Pattern Checker

A simple method to detect sentences generated by automatic generators is by brute force, comparing the patterns (string tokens) in a document with known patterns in generated documents. This approach was used internally by Springer in the earlier days to detect SCIgen generated documents.

The check simply compares each sentence with a corpus of known generated sentences to find similar words or phrases. Each similar bi-grams to four-grams is given 10 points, a similar phrase from five to nine words is given 50 points and more than 10 words phrase worth 100 points each. The total score is then compared with a threshold where 500-1000 points means it is a suspected generated document, while more than 1000 means it is surely generated.

We were able to obtain this program and modified it to discover the raw number of detected patterns for each sentence. The modified program scans the document and marks all the suspected patterns, then the sentences that contain those detected patterns are marked as automatically generated regardless of their length or structure. The result is shown in Table 4.6; since the program only focuses on SCIgen patterns so the test was only performed on SCIgen and SCIgen-Physics samples.

Table 4.6 shows that a simple pattern checker method still misses many generated sentences. Or in other words, this method does not detect many generated sentences and would be even more

[4] Intel Core I5 2.4 GHz with 16Gb Ram

impractical for the case of a small, partially generated document or a modified generator.

4.4.2 Traditional Machine Learning Techniques

To further evaluate the approach, R and Rtexttools package[Collingwood et al., 2013] are used; this package is used for supervised learning and includes different learning algorithms. The PCFG corpora $\mathscr{T}$ were converted to texts and separated into sets of three consecutive sentences. Since it is also required to have a sample corpus of genuinely written papers, other real corpora of the same size to the counterpart PCFG corpora $\mathscr{T}$ were chosen at random to be used as genuine-sample-corpora $\mathscr{D}$. The test corpus $\mathscr{C}$ was also split into sets of sentences.

So, in short, sentences from $\mathscr{D}$ are marked as "real" and sentences from $\mathscr{T}$ are marked as "generated", and then those sentences are used to train different types of classifier using a sentence-term-matrix. These classifiers are then in turn used to classify sentences form $\mathscr{C}$ to obtain the "generated" or "real" for each sentence.

It is understandable that for a truly impartial comparison, the information from the parse trees should also be given to the classifiers. However, transforming a parse tree to a feature vector is not a straightforward task and it may even be impossible in practice. If trees are added as a particular feature for learning algorithms, then one would have to define how to compute similarity for this particular feature and this is exactly what GSS is defining.

These machine learning techniques include:

- Glmnet [Friedman et al., 2010]: is an algorithm for estimation of generalized linear models with convex penalties. Their models include regression, two-class logistic regression and multinomial regression problems.
- Max entropy [5]: this package aims to obtain the maximal information entropy of distribution.
- SLDA [Myers et al., 1994]: calculates maximum likelihood estimate, exact and asymptotic confidence intervals.
- Boosting [Friedman et al., 2000]: Ada boost or adaptive boosting is a form of linear regression to minimize lost machine learning technique.
- Bagging or Bootstrap aggregating: is a model averaging approach which aims to improve accuracy, reduce variance and avoid overfitting.
- Tree or Decision Tree learning: aims to create a simple model that predicts the value of an input based on its variables.
- Random Forest [Liaw and Wiener, 2002]: is based on multiple classification trees, the final classification result is the most voted on result among those trees.

The results of these classifications are shown in Table 4.7. This table shows that conventional machine learning methods might not be appropriate to our need since the results vary from very bad (Glmnet, SLDA, Tree), where most of the sentences were marked as "generated," to mediocre (Max entropy, boosting, bagging random forest) where they marked about half of the genuinely written sentences as "generated."

For the GSS only and GSS system, 0.5 was used as a threshold to determine whether or not a sentence is automatically generated. As seen in Figure 4.1 with GSS only, this is not a good threshold

[5] http://www.logos.ic.i.u-tokyo.ac.jp/ tsuruoka/maxent/

	False positive rate		False negative rate		Precision	Recall
corpus size / algorithm	160	40 SCIgen	160	40 SCIgen	160	160
Glmnet	0.02	0.03	0.87	0.63	0.81	0.2
Maxentropy	0.13	0.14	0.62	0.45	0.66	0.48
SLDA	0.01	0.04	0.97	0.63	0.67	0.22
Boosting	0.5	0.14	0.11	0.49	0.55	0.79
Bagging	0.05	0.06	0.5	0.42	0.87	0.65
Random Forest	0.05	0.05	0.58	0.44	0.85	0.57
Tree	0.02	0.03	0.88	0.65	0.80	0.12
GSS only	0.95	0.73	0.007	0.015	0.41	0.993
GSS system	0.008	0.002	0.19	0.21	0.98	0.81

Table 4.7: False positive rate, false negative rate as well as precision and recall of different methods with different corpus type

for a single sentence; however, if the context is taken into account as in GSS system (Figure 4.3), a very promising result is obtained with very few genuinely sentences marked as "generated" (for corpus size 160 there were less than 200 sentences marked as automatically generated out of more than 22k genuinely written sentences) but still catches a good number of automatically generated ones.

Furthermore, to verify the possibility of detecting a modified generator where only the terminal terms or keywords were changed, Physgen which is only a version of SCIgen with all the "hot keywords" switched from computer science to physic ones is used. For this test, a corpus of 40 SCIgen papers is used to try to detect physgen sentences among 100 Physgen papers and 100 genuine papers.

As before, a genuine-sample-corpus of 40 genuine papers is also used to aid machine learning techniques. The results are shown in the "40 SCIgen" column of Table 4.7. As suspected, using GSS only it was able to catch most of the sentences from Physgen with only samples from SCIgen (0.015 false negative rate); even if the context and Jaccard filter are used, the GSS system is still able to find 80% of them. This suggests that the GSS system would also be effective against cases of newly modified versions of existing generators.

4.4.3 Performance Evaluation

To accurately evaluate the performance of the system, an evaluation was performed as follows. 10 genuine documents are used, and each is injected with 50 different sentences from SCIgen in two random paragraphs of 25 sentences each as seen in Figure 4.4.

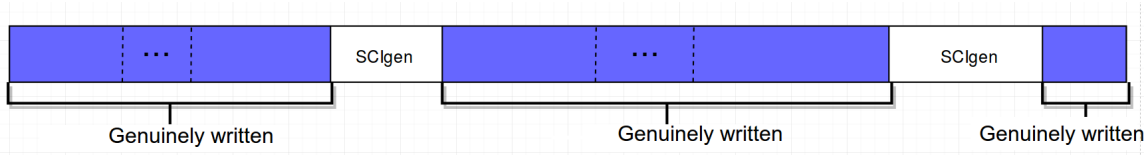


Figure 4.4: Example structure of a genuinely written document injected with automatically generated sentences.

These documents are then run through the GSS system to observe how many automatically generated sentences would be picked up in this situation. The result of this test is quite encouraging since for each document, on average 33.57 out of 50 automatically generated sentences are detected. In the worst case, the system could detect 27 sentences, and closer examination of this case reveals that one of the SCIgen paragraphs contains some short, un-structured mathematical formulas. This once again confirms our assumption that the system can detect small paragraphs of automatically generated text inside a genuinely written document.

Furthermore, we test our system on the paper by Navin Kabra that was mentioned earlier. By our count, the paper contains 135 sentences in total, and there are 44 sentences that were written by human, including 16 direct copies from a movie script. Using our newly developed GSS system, we could point out 65 in 91 generated sentences with only 7 false positive cases. This is a test for a real-life situation that can happen again in the future. Other cases of known partially generated papers are also tested. But without a confirmation from the authors, it is impossible to know how many sentences in those papers are genuinely written. Nevertheless, by our test, the paper in Example 1.1 may have 91 automatically generated sentences in its 152 sentences while the paper Figure A.2 has 204 sentences and 122 are marked as automatically generated.

4.5 Summary

So, in this chapter we have shown our GSS system which can detect sentences from known PCFG generators with sufficient samples, which has an 80% positive detection rate and less than a 1% false detection rate. Furthermore, the system has been bench marked against some well-known machine learning techniques to demonstrate that it can provide the best results. The possibility of detecting a modified version of current generators was also verified with great success.

However, using the system against generators which use other techniques, such as Markov model or RNN is impractical and would call for some other approaches as in Chapter 3. Thus, we broadened the investigation with an aim to detect “meaning-less” scientific papers based on word embedding in the next chapter.

Despite all of that, the system still needs many improvements. One obvious change would be the implementation of a lemmatization before the parsing step, as this would promise a better result. Otherwise, it could also be improved on to work as plagiarism detection where words are changed from the original sentences. Nevertheless, current results are still quite promising for the desired goal.

Chapter 5

Detecting Generated Text Without Samples

Previous chapters 3 and 4 have shown that documents or even segments from automatic generators can be detected. However, these methods are inadequate when facing a new generator where it is impossible to accumulate a reasonable number of samples. The aim of this chapter is to attempt different methods to detect automatically generated texts without any hint from a sample corpus or in other words, detect automatically generated texts using characteristics of genuinely written ones. In particular, we start from simple characteristics such as the number of words inside a fixed window and then move to try to detect words with "bizarre" neighborhood compared to what can be normally obtained. First in section 5.1, we explore the possibility of using the growth of the vocabulary of an article or a small section as a method to distinguish an automatically generated document from a genuine one. Further, Section 5.2 gives some background work of word embedding with Word2Vec and how it can be used in our context. Based on that, Section 5.3 discusses how the common neighborhood of a word can be used against automatically generated documents with nonsensical phrases.

5.1 Vocabulary Growth

5.1.1 Vocabulary Growth as a Classification method

Vocabulary growth has been mostly researched for the link between development of one's age, gender and the respective size of his or her vocabulary [Huttenlocher et al., 1991]. Nevertheless, in our particular case where vocabulary growth rate is used to distinguish different types of texts (generated/genuine), we did not find much prior research beside [Labbé and Labbé, 2014]. In this research, the authors compared the vocabularies of several famous writer's works of unequal length. From there, they show that there is a correlation between the length and the size of the vocabulary of each writer. However, contrary to common belief, the vocabulary of William Shakespeare is not unusually "rich" but is only within the average of his contemporaries.

This approach is based on the fact that current generators (that use Probabilistic Context Free Grammar) generated keywords at random. For example, a sentence generated by SCIgen could be obtained from this rule: "Many SCI_PEOPLE would agree that, had it not been for SCI_THING, the SCI_ACT might never have occurred." In this sentence, SCI_PEOPLE can be

chosen at random from the following set {leading analysts, experts, analysts, scholars, information theorists, statisticians, computational biologists, etc. }; SCI_THING as in {IPv4, IPv6, DHTs, robots, agents, Markov models, the memory bus, SMPs, kernels, suffix trees, spreadsheets, etc.} and the same for SCI_ACT. Even though this method produces a valid sentence, over time the accumulation of all those random keywords might be different from a human written scientific paper which focuses on a specific topic. That means certain keywords should be repeated from time to time.

One might also think about the opposite, where even though the keywords are chosen at random but the vocabulary richness is not high, only some word types are available to be chosen over and over again as well as a limited number of sentence structure to choose from, thus resulting in a smaller vocabulary size than normal.

So, our hypotheses are that: consider a long, fully generated PCFG text, the vocabulary might be less diverse than a genuinely written document of the same length because the richness of the generators can not be compared to human vocabulary. However, in a short, confined "slide" of text, the opposite might be true. Since generally, genuinely written texts in a small section are focused in only a particular topic with repetitive words while automatically generated ones are more random.

To understand this proposal, we present some statistical information that we gathered in Section 5.1.2. From this information, Section 5.1.3 tries to reapply the finding on a test corpus to classify documents as generated or not.

5.1.2 Statistical Information about Vocabulary Growth

We counted the number of different word-types in our corpus of genuine papers and automatically generated papers to see the differences in the growth of the vocabulary of each type of paper.

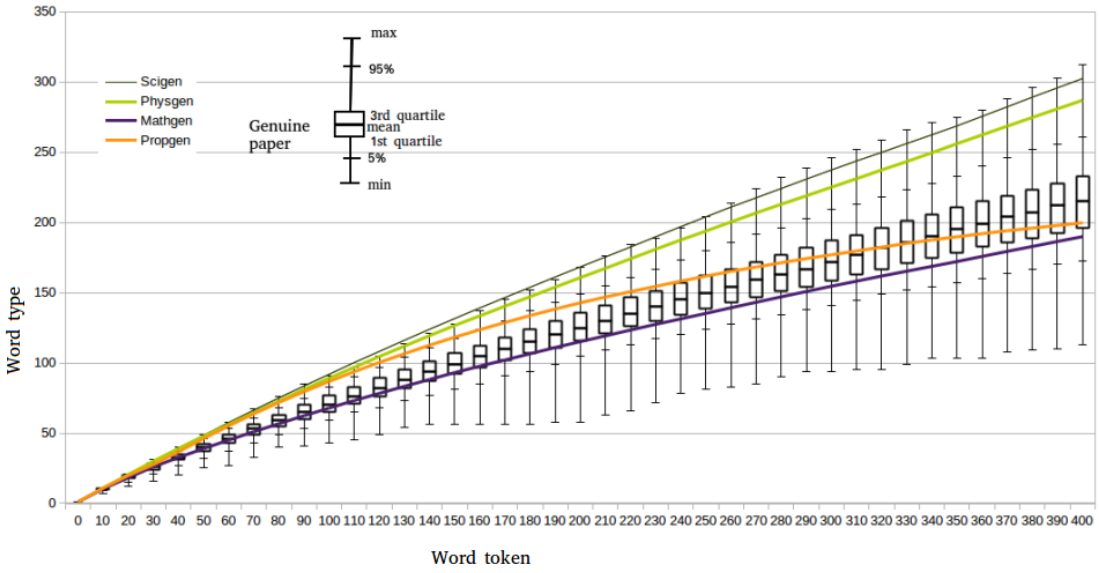


Figure 5.1: The growth of the vocabulary overtime of Genuine paper and the average growth of generated paper

Figure 5.1 shows the difference of the vocabulary growth in average of 400 real papers along with the average of 400 documents for each type of generators. Even though there is a large variation of vocabulary size from a genuine paper, we can still see there is a distinction from a genuine paper compared to generated ones starting at about word token index 50. The distinction in vocabulary size grows more significantly later on in the cases of SCIgen and Physgen. However, for Mathgen and Propgen, they seem to grow at a slower rate and then somewhat under-grow compared to what normally occurs. This phenomenon might result from the fact that the choice for words as well as sentences in PropGen and MathGen being not as varied as their counter-parts. In fact, we counted the total number of different words (excluding stop-words and non-dictionary words) that appear in a single paper from a corpus of 500 Propgen generated ones. This number always come out between 345 to 360 words, does not matter what length the paper is.

We also believe that words in a particular section should be repeated at a particular rate. Thus, we investigated the richness of the vocabulary inside a window of text, not just the paper as a whole.

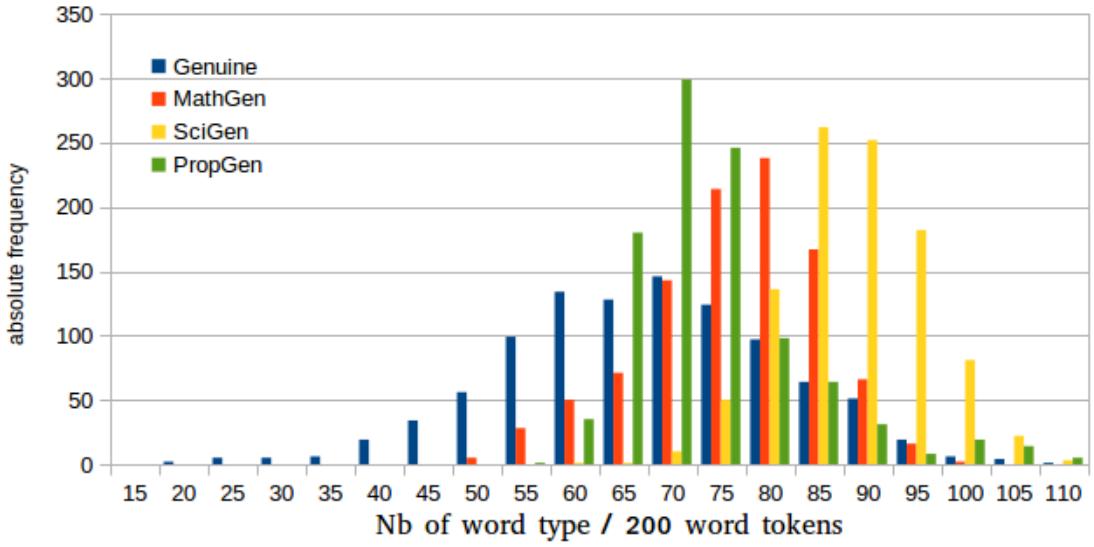


Figure 5.2: Vocabulary richness inside a window of 200 word token from 400 documents for each type.

Figure 5.2 shows that in a generated paper, a 200-word token window appears to have more different word types than what normally appear in a window of genuine text. From there, we would want to see in a genuine as well as generated text, what percentage of the 200-word token window would fall outside of the "normal" area (that we set at more than 65 different words/window of 200 words). Figure 5.3 shows that for a generated text, most, if not all the window, would have more than what we consider normal, while for genuine text, this percentage is much smaller. This could also be used to determine characteristics of a generated paper.

Based on this preliminary information, the next section will further explore the possibility of

using the characteristics of the vocabulary size as a method to classify automatically generated documents.

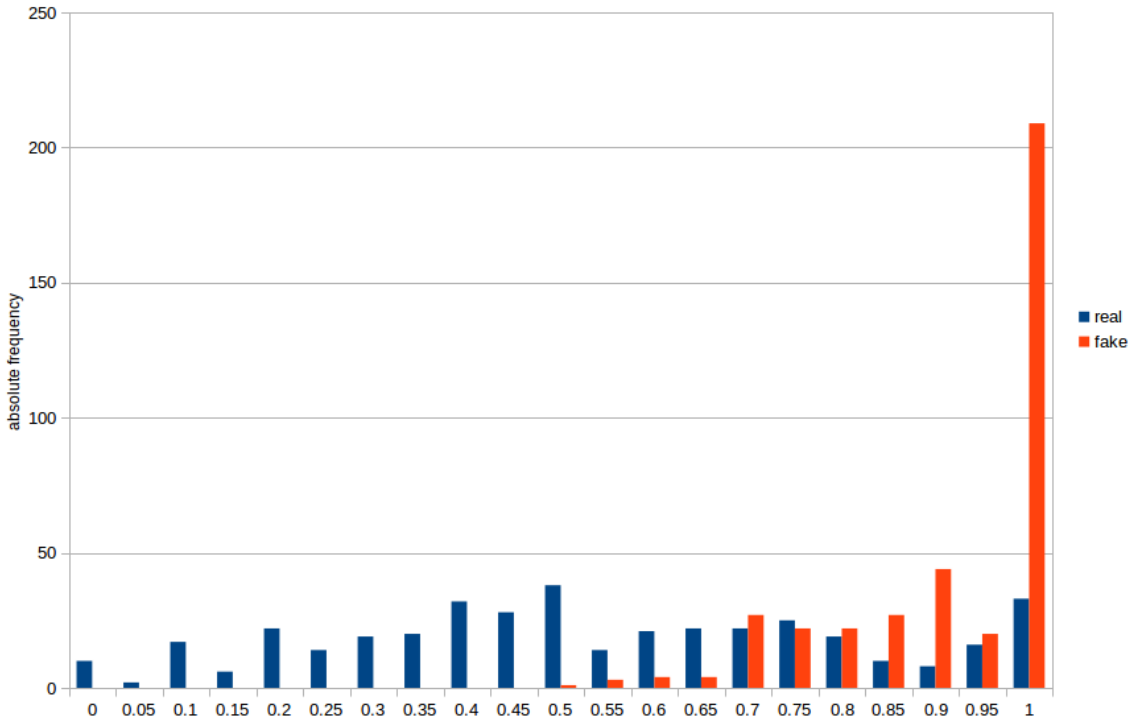


Figure 5.3: Percentage of windows that have more than 65 different word types.

5.1.3 Preliminary Test and Results Using Vocabulary Growth

We have implemented our proposal in Java based on what we gathered in the statistical information. In the implementation, we have set the following process which is the combination of two methods for each document under test. These hard thresholds are chosen based on the previous section to get the best distinction:

- Count the total number of different word types (excluding stop-words and non-dictionary words). If this total number of word types is less than 360 while the current paper under test has more than 2000 words (a typical length of at least two pages), this is abnormally low and it will be considered as a PropGen paper.

- Count the size of the vocabulary of every 10-word token starting from word token number 50 to word token number 250 (thus 20 windows).

  - If there are more than 10 windows with the vocabulary size falling outside the "normal" range (the mean ± STDEV); we mark it as automatically generated.

- Split the paper into windows of 200-word token and count the size of the vocabulary inside these windows.

  - If more than half of the windows have more than 65 word types; we also consider it as an automatically generated one.

- Otherwise, it is considered as genuine.

Result: Using our tool, we tried to scan 400 genuine papers in different fields along with 400 generated papers from four known generators. The results are as follow.

Out of 400 genuine papers:

- 56 false positives: 39 were identified as fake, 17 as PropGen.

- 344 true negatives.

- Subsequently False positive rate: 0.14

Table 5.1 shows the results of 400 generated papers:

	SciGen	PhysGen	MathGen	PropGen	Total
True Positive	100	99	18	100	317
False Negative	0	1	82	0	83
Recall	1	0.99	0.18	1	0.79

Table 5.1: Scanning results for 400 generated papers using vocabulary growth.

We have obtained perfect result for SciGen, PropGen, as well as close to perfect for PhysGen. However for MathGen the result is not so good, this might be because the vocabulary richness of MathGen is not comparable to SciGen or PhysGen but not as low as PropGen. In another word, there are less choices for sentences and keywords in MathGen compare to the others however this choice is not as low as PropGen and thus somewhat overlap with genuinely written papers.

On the other hand, the number of false positives is considerable since the written language can largely vary depending on the topic or the author. To limit this number, we investigate a different approach based on the neighborhood of words.

5.2 Using Word Embedding

Word embedding is a technique where words or phrases are mapped to vectors of real numbers. These vectors can then be used in different applications such as information retrieval [Ganguly et al., 2015, Palangi et al., 2016], sentiment analysis [Tang et al., 2014], machine translation [Zou et al., 2013], question answering [Ganguly et al., 2015] and much more.

Our hypothesis for this section is that words from a same document must be somewhat similar to each other so the vectors that represent them should also be similar to each other. Thus, this section presents Glove and Word2Vec as methods to obtain vector representations for words, and then Word2Vec is focused on as the most used method and tested with our corpora.

5.2.1 Word2Vec

Word2Vec (W2V) is a program that was developed by Tomas Mikolov et al. at Google [Mikolov et al., 2013d, Mikolov et al., 2013c, Mikolov et al., 2013a]. It is used to create vectors to represent linguistic context of words using a shallow two layers neural network. The aim of W2V is to learn vectors that represent words inside a particular corpus, and words with similar representation (vector direction) should be somewhat similar in meaning and using these vectors, relations between words can be revealed (Figure5.4).

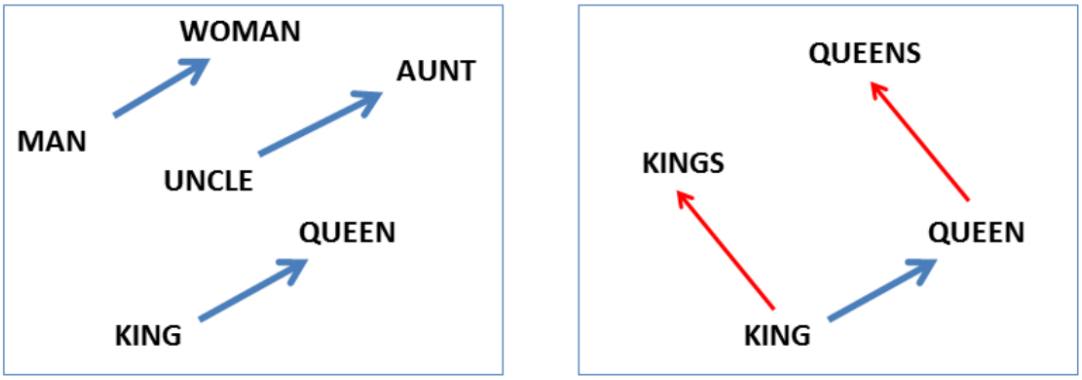


Figure 5.4: "Left panel shows vector offsets for three word pairs illustrating the gender relation. Right panel shows a different projection, and the singular/plural relation for two words. In high-dimensional space, multiple relations can be embedded for a single word" (From [Mikolov et al., 2013d])

In [Mikolov et al., 2013a], the authors introduce the continuous skip-gram model to be used for learning high quality word vectors for millions of types from a corpus of billions of tokens. Two log-linear models were proposed and tested, namely continuous bag-of-words mode (CBOW) and continuous skip-gram model (Figure 5.5). In CBOW, the input to the model are windows that surrounding words and the output layer output can be thought of the task as "predicting the word given its context." While Skip-gram is a mirror image of CBOW, where the input layer is the target words, and the output layer of the neural network is replicated multiple times to accommodate the chosen number of context words inside a window C or, in other words, "predicting the context given a word." To verify the results, the authors have shown that using well trained vectors, it is possible to deduce similarities between word pairs such as France is to Paris is similar to Germany is to Berlin. From there, simple algebraic operations can be used with the vector representation of words to understand word relationships like $vector(\text{“}biggest'') - vector(\text{“}big'') + vector(\text{“}small'') \sim vector(\text{“}smallest'')$. Furthermore, these models were tested using several LDC corporal [Mikolov et al., 2011] with 320 millions word tokens and 82 thousands word types vocabulary, words are represented in a 640 dimension vector space. It was shown that CBOW works well on both syntactic tasks and semantic tasks while skip-gram, despite a slightly worst performance on the syntactic tasks than CBOW, is much better on the semantic part of the test. From there, these models were used to tackle The Microsoft Sentence Completion Challenge [Zweig and Burges, 2011] which has 1040 sentences,

each missing a word and the goal is to propose the missing word that is the most coherent with the rest of the sentence. The Skip gram model was trained on a provided 50M words corpus with 640 dimension vector space, the obtained result did not compare well to an average LSA similarity. However, combining the skip-gram model with RNNLMs [Mikolov, 2012], the authors were able to outperform state-of-the-art results.

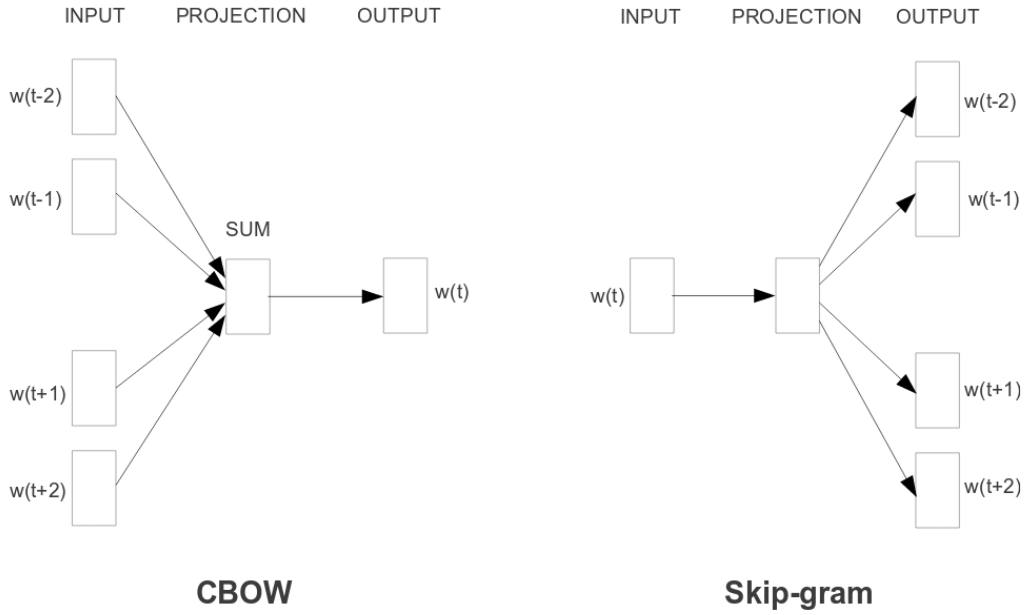


Figure 5.5: "New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word." (From [Mikolov et al., 2013a])

Following the previous results, Tomas et al. continued to improve the skip-gram model with [Mikolov et al., 2013c]. In this work, the authors show that by subsampling frequent words, the process can be significantly sped up with more regular word representations. The idea of subsampling is that frequent words such as stop-words (the, a, in, etc.) provide much less information value compared to the rare words. Thus, a subsampling probability $P_{(w_i)}$ is introduced to decide if the word w_i should be discarded

$$P_{(w_i)} = 1 - \sqrt{\frac{t}{f(w_i)}}$$

Where $f(w_i)$ is the frequency of word w_i and t is a chosen threshold. This formula greatly cuts out words that have a frequency greater than t while still preserving the ranking of the frequencies. To show that this subsampling process considerably accelerates the learning time and also significantly improves the quality of learned vectors, the authors have chosen several methods to be

compared with subsampling in the analogical reasoning task [Mikolov et al., 2013a]. The aim of this task is to demonstrate connection between words using simple algebraic on their vectors representation such as the connection Germany-Berlin and France-Paris or big-biggest and small-smallest as presented before. And the result supports the hypothesis that subsampling both improves the training speed as well as produces more accurate representation vectors.

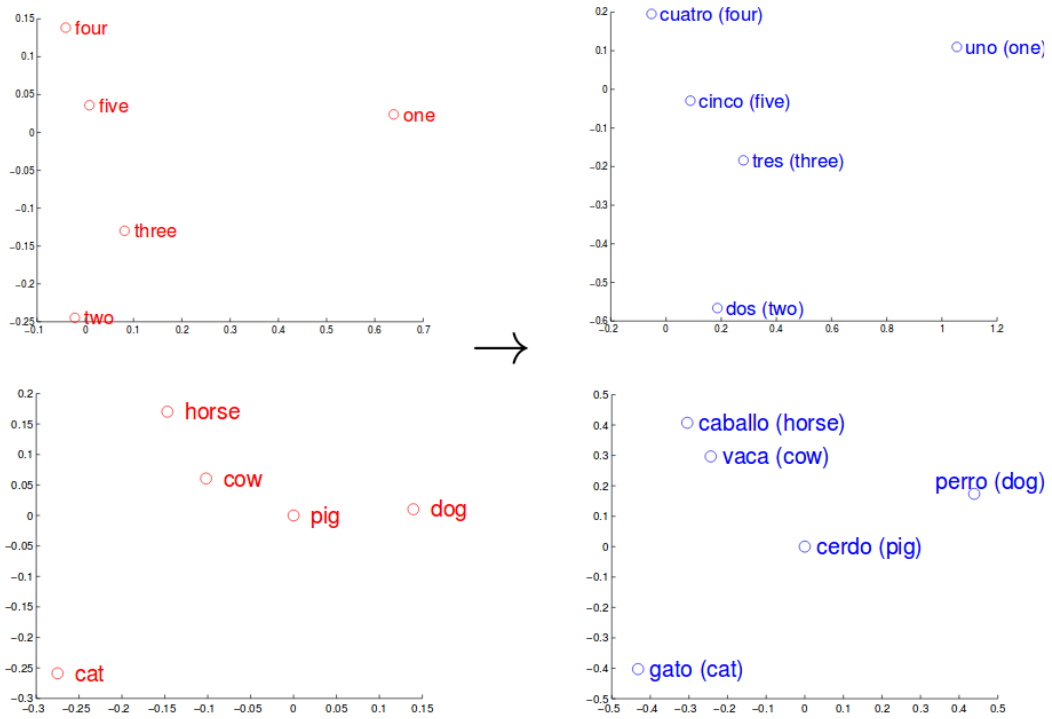


Figure 5.6: "Distributed word vector representations of numbers and animals in English (left) and Spanish (right). The five vectors in each language were projected down to two dimensions using PCA, and then manually rotated to accentuate their similarity. It can be seen that these concepts have similar geometric arrangements in both spaces, suggesting that it is possible to learn an accurate linear mapping from one space to another."(From [Mikolov et al., 2013b])

Word2Vec has been adopted in many different natural language processing tasks such as machine translation [Mikolov et al., 2013b]. The authors propose a method to automatically build or expand dictionaries and phrase tables by learning language structures and mapping between vector spaces of languages. The linear relationship between languages as visualized in Figure 5.6 shows that word vectors for English numbers somewhat correspond to their counterpart in Spanish. Thus, if a transformation matrix can be deduced from the relationship between one to five from English to Spanish, it can be used to translate other numbers to Spanish. This transformation matrix can in fact be learned by solving an optimization problem with stochastic gradient descent. And despite

its simplicity, this linear transformation works quite well as demonstrated in their experiments. In particular, the authors performed several tests, comparing the translation of a thousand words from a source language to a destination language by learning with the five thousand most frequent words in the source language and was able to obtain about 50% accuracy. Then the experiment was scaled up with a larger English-Spanish dataset with several billion words (Google News datasets). The same process of using the five thousand most frequent words to construct the dictionary is repeated and the result shows that the method still has a reasonably high precision at 60% and even more surprising is the fact that some words were translated correctly even though they are quite unrelated to the most frequent dictionary. Then the authors expanded the example with even more languages namely English-Czech, English-Vietnamese and show that it is also possible to correct dictionary errors. This was achieved by computing the distance between the word vectors of translation given by their system and the existing dictionary entries. If there is a strong disagreement between the two translations, then there might be some error in either of them.

5.2.2 Gloval Vectors - GloVe

In this work, [Pennington et al., 2014] analyze the model properties to produce linear direction of meaning (as space vector for *king - queen = man - woman*) and propose a method to produce word vectors with meaningful structure. To do such a task, the authors present the GloVe model for global vectors. In this model, words co-occurrence probabilities are calculated.

Specifically, for a word i with a word-to-word co-occurrence matrix X where X_{ij} is the number of times word j appears in the context of word i and $X_i = \sum_k X_{ik}$ is the number of times any words appears in the context of i. Then $P_{ij} = \frac{X_{ij}}{X_i}$ is the probability that word j appears in the context of word i. The authors show that the co-occurrence probability can be used to distinguish a relevant word pair from an irrelevant word pair. From this argument, they note that word vector learning should start with the ratios of co-occurrence probability rather than the word probabilities themselves. Then, using this information, equations are developed to establish a model for log-bilinear regression unsupervised learning.

To validate this model, a corpus with 6 billion tokens was created from a Wikipedia dump and Gigaword 5[1]. The representation vectors for 400,000 most frequent words in this corpus are learned using Word2Vec and GloVe. The authors compare the accuracy on different tasks with different parameters and conclude that the GloVe model is useful in downstream NLP tasks such as word analogy, word similarity and named entity recognition.

5.2.3 Implementation with Word2Vec

We believe that words inside a section or a window should be somewhat related to each other. However for the case of PCFG, again because the words are chosen at random, they wont be exactly correlated. Thus we proposed to compare the most frequent nouns in a window to observe how close they are compared to one another.

Recent trends are quite heavily focused on working with Word2Vec so we decided to follow and test our hypothesis using W2V. We trained our W2V vectors based on a text corpus from *Lecture Notes in Computer Science* series that we were able to obtain from Springer which has more than 150 million word tokens (about 1.3 GB of text) and more than 250 thousand word types feature vectors (LNCS corpus). This corpus was then used to learn feature vectors with both CBOW and skipgram models, each using a window of four, 100 dimension vector space and negative learning 25. The learning results were then translated from binary to a text file for easier processing, to

[1] https://catalog.ldc.upenn.edu/ldc2011t07

further reduce the processing time, all non dictionary, non noun words are removed. In the end, we were able to obtain 35k noun feature vectors.

From then, these vectors are used to classify different types of scientific papers from different corpora that were presented in Chapter 4 . For each paper under test, it was split into windows of 200 word tokens; from each of these windows, the ten most frequent non stop-word nouns were selected and the average pair-wise cosine similarity (APS) or Euclidean distance between them are computed.

Let F_A be the set of n most frequent nouns from each windows of 200 word tokens, each noun is represented by a vector $\vec{V}$ then

Average pair-wise cosine similarity of the windows: $APS = \frac{\sum_{x,y \in F_A} cos(\vec{V_x}, \vec{V_y})}{n}$

while Average pairwise Euclidean distance of the windows: $APE = \frac{\sum_{x,y \in F_A} euclideandistance(\vec{V_x}, \vec{V_y})}{n}$

Figures 5.7 and 5.8 show the pair wise cosine similarity for the test corpus respectively with representation vectors that were learned using the CBOW or the skip gram model. Similarly, Figures 5.9 and 5.10 show the pairwise Euclidean distance for the test corpus with different representation vectors.

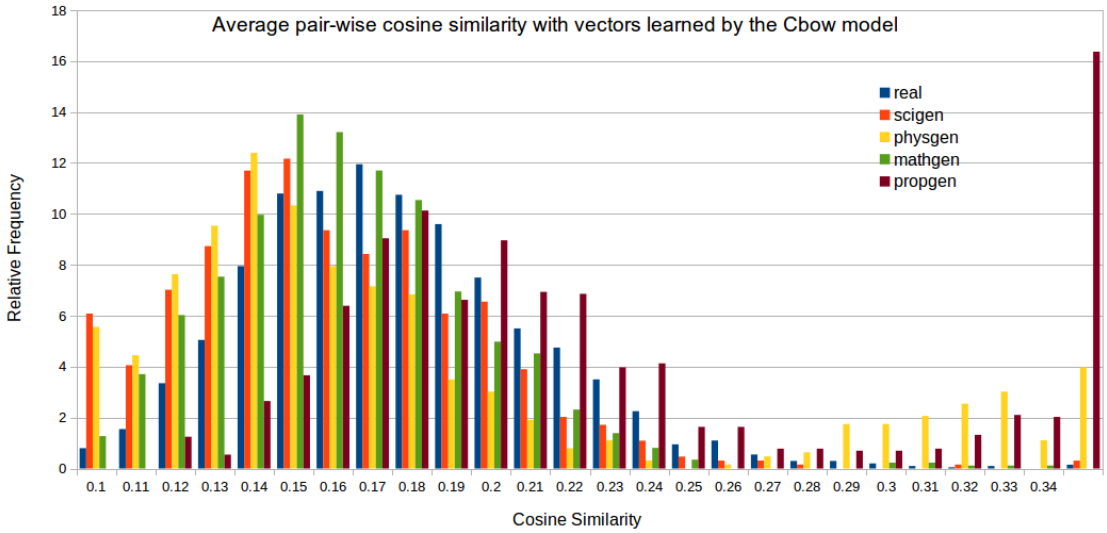


Figure 5.7: Histogram of average pairwise cosine similarity for nouns from a window of 200 word tokens in different types of papers using vector representation that were learned with the CBOW model.

Looking at these Figures, it is hard to determine a clear method to confidently classify a document as genuine or automatically generated. Except for Propgen with CBOW feature vectors, both with cosine similarity and Euclidean distance, Propgen generated papers seem to have quite a different distribution, which might be caused by the complex noun where it is impossible to find a matching noun in our learning corpus. This discourages us from further testing with Word2Vec, however we still hold our belief that words can be described by their surrounding windows, thus the next section will explore the possibility of classifying text based on the neighborhood of words.

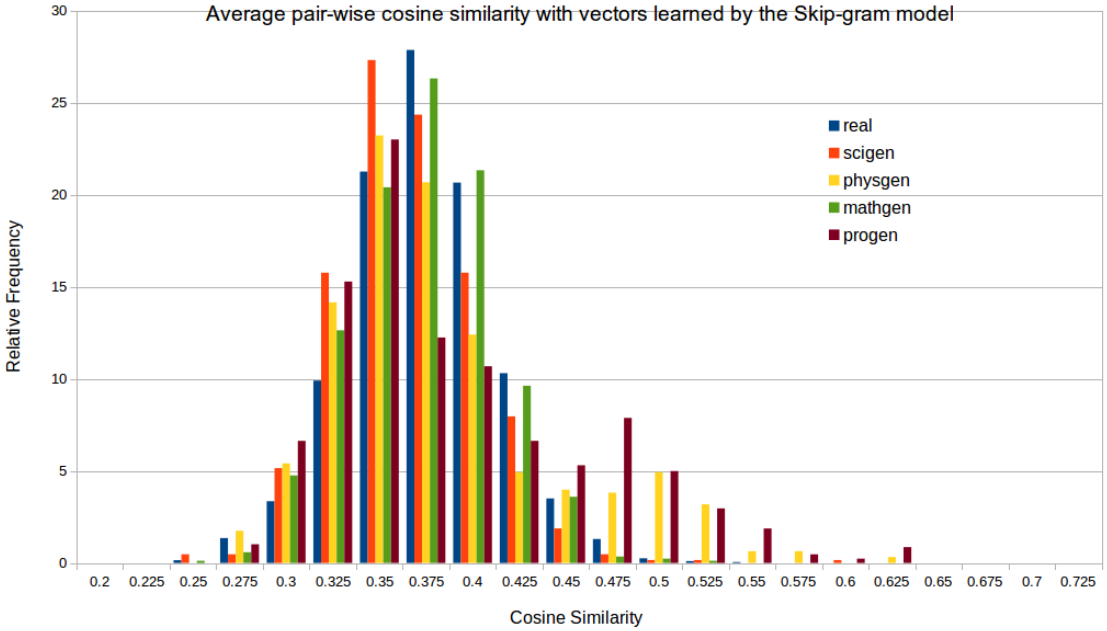


Figure 5.8: Histogram of average pairwise cosine similarity for nouns from a window of 200 word tokens in different types of papers using vector representation that were learned with the skip-gram model.

5.3 Classification Based on Word's Neighborhood

"You shall know a word by the company it keeps" [Firth, 1957], and words in a small section should be somewhat related or similar to each other. We investigated these claims in the following section. The hypothesis of this approach is that words can be characterized by the surrounding neighbors and different styles of text should result in different word's neighborhoods.

5.3.1 Implementation Method for using Word's Neighborhood

The previous LNCS corpus is again used in this method, it was normalized, and then from each word, a window is extended to either side with a certain size or until the beginning/end of the sentence is reached. This process should encapsulate all the neighbors and then information is kept in a list with the absolute number of co-occurrence between the word and its neighbor. After all the words have been processed, the absolute frequency of co-occurrence is translated to relative frequency or co-occurrence rate. This should ensure that the length of the corpus does not heavily affect the outcome. In the end, each word is represented by a vector of a certain size of its most popular neighbors. These vectors are then used to calculate the differences between them and vectors in a paper under test to determine whether the vectors under test have an abnormal neighborhood. The details of these processes are presented in the next section.

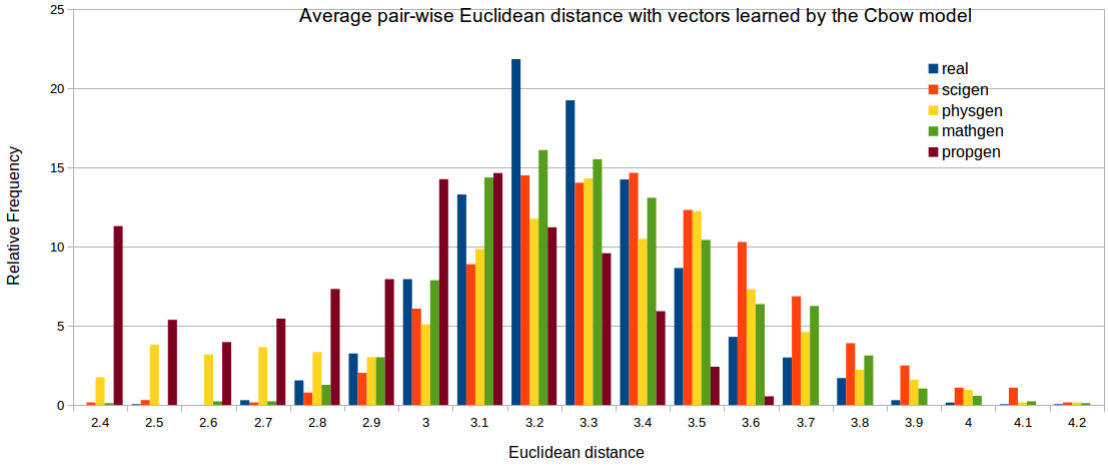


Figure 5.9: Histogram of average pairwise Euclidean distance for nouns from a window of 200 word tokens in different types of papers using vector representation that were learned with the CBOW model.

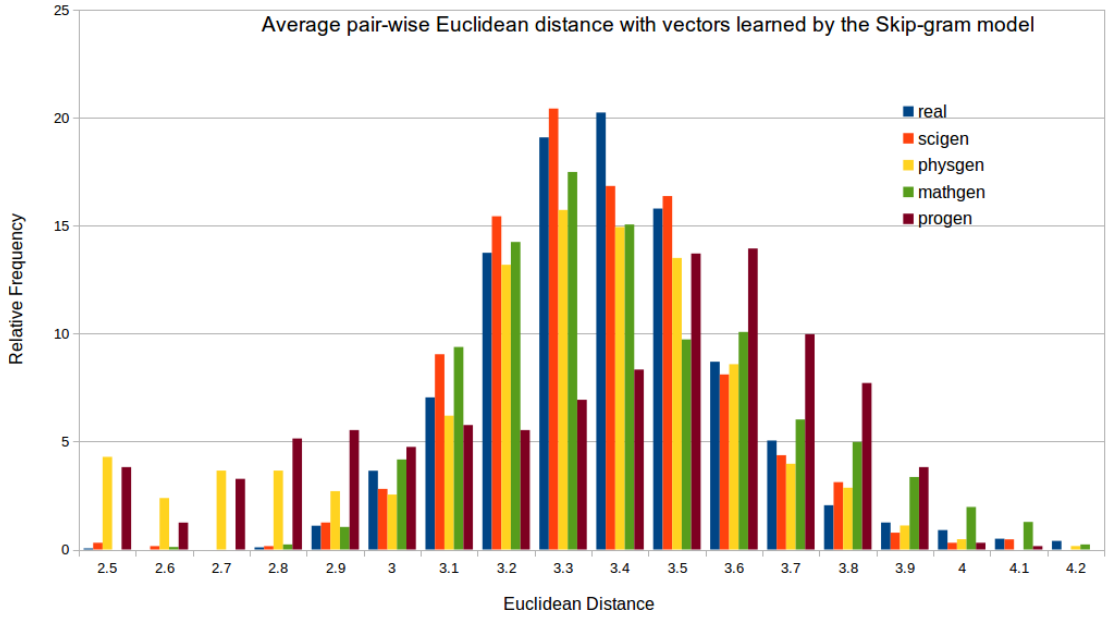


Figure 5.10: Histogram of average pairwise Euclidean distance for nouns from a window of 200 word tokens in different types of papers using vector representation that were learned with the skip-gram model.

5.3.2 Experimental Process

As the beginning of the experiment, we realized that there are multiple parameters that will affect the results, namely the size of the window that would be considered as the neighborhood of the words, how many frequent neighbors should be kept inside the neighborhood, and how to calculate the similarity of the neighborhood.

Choosing the Number of Frequent Neighbors

Firstly, we would like to know how many common words the neighborhood has in common. Thus, different sizes of the neighborhood will be tested. Specifically, in each document, the most 50 frequent non-stop-word nouns are discovered, then all the words in the sentences that contain the frequent words are recorded. From these lists of recorded neighborhoods, the most frequent neighbors are compared to see on average how many neighbors do words from a generated or genuinely written document have in common with a learned neighborhood from genuinely written ones? Jaccard similarity was used to demonstrate how much similarity they do share; the higher this value, the more neighbors they have in common.

Let F_A be the set of n most frequent keywords in a document A, for each keyword $x \in F_A$ is represented by a set $N_A(x)$ of m of its most frequent co-occurrence neighbors. F_C be the set of all words that were learned by the corpus, each word $y \in F_C$ is also represented by a set of $N_C(y)$ of m of its most frequent co-occurrence neighbors. Then the average Jaccard Similarity between A and the learned corpus C is:

$$JaccardSim_{A,C} = \frac{\sum_{w \in F_A} \frac{N_A(w) \cap N_C(w)}{N_A(w) \cup N_C(w)}}{n}$$

Figures 5.11, 5.12 and 5.13 show that as we expected, in general automatically generated documents seem to share less common neighbors with the learned neighborhood compared to genuinely written ones. It is also understandable that as the size of the neighborhood is increased, the average Jaccard similarity decreases since the neighbors of words from a document under test is quite fixed, and even if the number of most frequent neighbors is increased, there is a limited number of them. However, for a word's neighborhood learned from a big corpus, the neighborhood would be more diverse and would likely be expanded when the number of most frequent neighbors is considered. Nonetheless, with 20 most frequent neighbors, both generated and genuine documents shared a reasonable number of neighbors with the learned neighborhood which would provide a better result while using different similarity calculations. So, from now on, we would work with 20 of the most popular neighbors for each word.

Choosing the Size of the Neighborhood Window

The next step is to decide what is considered as the neighborhood or how many of the adjacent words would be connected to the word under test. To do this, Figures 5.14, 5.15, along with Figure 5.11 show different window size of adjacent words to be considered as neighbors. It can be seen that reducing the window size to two adjacent words affects Propgren the most, sometimes even half the Jaccard similarity while genuinely written documents do not differ much. The same phenomenon occurs for other types of generators where the Jaccard similarity slightly decrease whenever the size of the neighborhood window is decreased. Given these results, we decided that each word will be represented by its 20 most popular neighbors inside a window of five words.

To further the experiment, we would like to test different methods to calculate distance/similarity given that each word is represented by a vector of neighbors as stated earlier on.

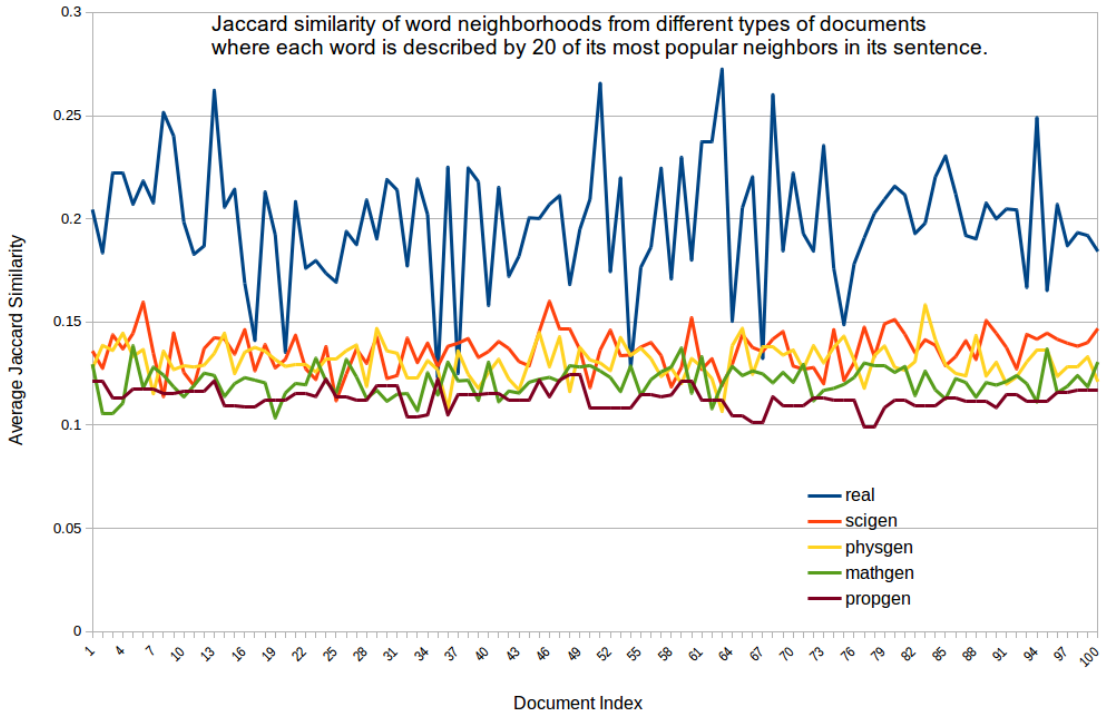


Figure 5.11: Jaccard similarity of frequent word neighborhoods from different types of documents where each word is described by 20 of its most popular neighbors in its sentence.

Choosing the Similarity Calculation

In this section, some popular methods to calculate the similarity/distant between a word's neighborhood are presented. Again, in each document A, $n = 50$ most frequent non-stop-word words are selected as set F_A, and then for each instance $w \in F_A$ of these words, a neighborhood window of five words is extended to either side to collect all the relative frequency of co-occurrence neighbors $N_{A(w)}$ where $f(x \in N_{A(w)})$ is the relative frequency of time where word x appears in the neighborhood of word w. Using these lists of neighbors, the 20 most frequent neighbors and their relative frequency of co-occurrence to each word w in F_A will be used to calculate these distance/similarity with the learned set F_C from corpus where each $w \in F_C$ is also represented by a set of relative frequency of co-occurrence neighbors $N_C(y)$.

Cosine Similarity for each word $w \in F_A$ to the same word in learned corpus C:

$$CosineSim_{(A(w),C(w))} = \frac{\sum_{x \in N_{A(w)} \cup N_{C(w)}} N_{A(w)}x \times N_C(w)x}{\sqrt{\sum_{x \in N_{A(w)}} N_{A(w)}x^2} \times \sqrt{\sum_{x \in N_{C(w)}} N_{C(w)}x^2}}$$

where $N_{A(w)}(x) = \begin{cases} f(x \in N_{A(w)}), & \text{if } x \in N_{A(w)} \\ 0, & \text{otherwise} \end{cases}$

In a simpler term, for each word in the set of most frequent keywords from the document under test A, its combined popular neighborhoods from A and from the learned corpus C are gathered as

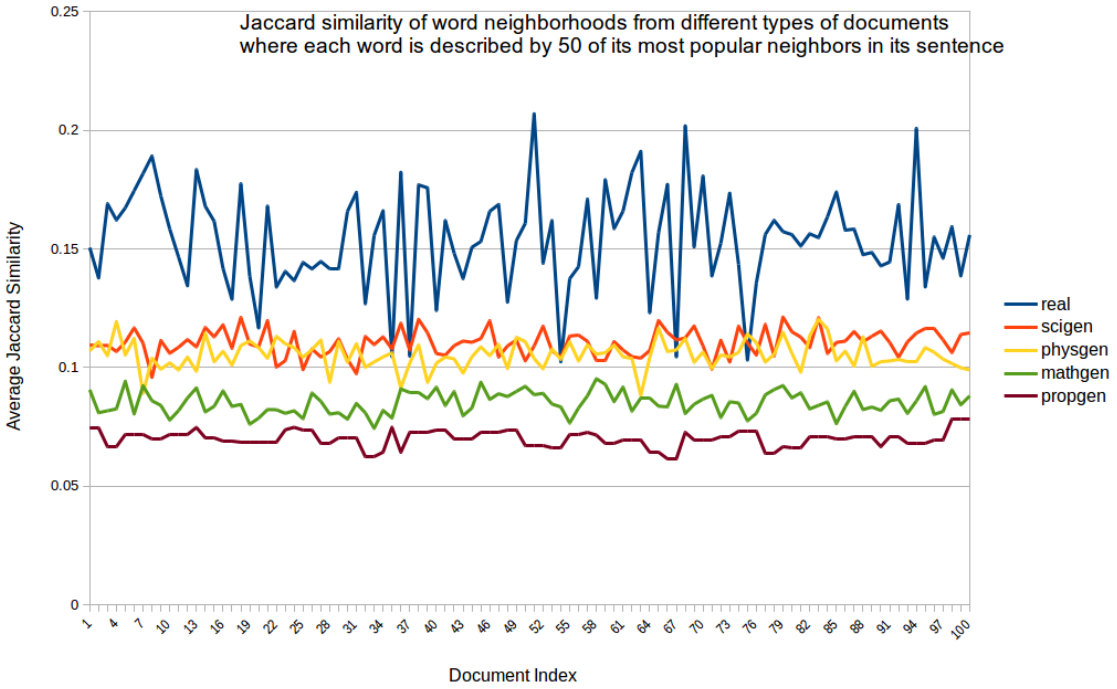


Figure 5.12: Jaccard similarity of frequent words neighborhood from different types of documents where each word is described by 50 of its most popular neighbors in its sentence.

$N_{A(w)} \cup N_{C(w)}$. For each instance of neighbor x in the combined neighborhood, the co-occurrence relative frequency $f(x \in N_{A(w)})$ or $f(x \in N_{C(w)})$ are used to calculate the cosine similarity. If x is not present in either of the neighborhoods, the co-occurrence rate is set as 0.

Figure 5.16 shows the distribution cosine similarity of all the frequent words from the test corpus. It can be seen that words from auto generated documents tend to have lower cosine similarity compared to words coming from genuinely written documents. In particular, about 60% of keywords from genuinely written document have 0.5 or higher cosine similarity while only 40% of keywords from Mathgen and Propgen and 25% for Scigen, Physgen. Furthermore, a much bigger percentage of keywords from genuinely written documents have a decisively high cosine similarly of 0.8 or 0.9 which make them have a much higher probability of being in a proper context. However, the distributions are spread quite wide and it would be difficult to create a specific threshold for classification.

The possibility of using Euclidean distance is also tested in Figure 5.17. However, this does not seem to be the best method, despite the fact that keywords from genuinely written documents do have smaller distance to the learned neighborhood but the distances from other type of documents are also quite clustered together and would be hard to precisely separate them. Euclidean distance for a set of F_A to learned corpus:

$$EuclideanDist_{(A_{(w)},C_{(w)})} = \sqrt{\sum_{x \in N_{A(w)} \cup N_{C(w)}} (N_{A(w)}x - N_{C(w)}x)^2}$$

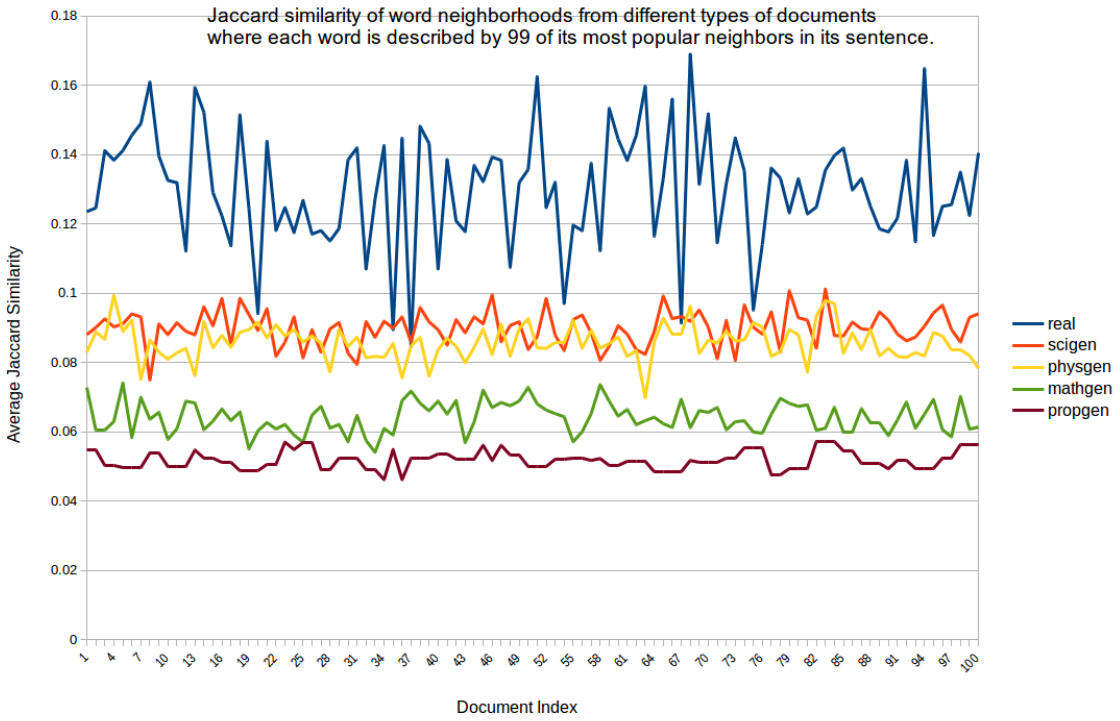


Figure 5.13: Jaccard similarity of frequent words neighborhood from different types of documents where each word is described by 99 of its most popular neighbors in its sentence.

where $N_{A(w)}(x) = \begin{cases} f(x \in N_{A(w)}), & \text{if } x \in N_{A(w)} \\ 0, & \text{otherwise} \end{cases}$

Furthermore, Figure 5.18 shows the distribution of relative frequency when using textual distance [Labbé and Labbé, 2013] between a word's neighborhoods from different types of documents. As seen, the majority of keywords from genuinely written documents closely resemble the word's neighborhood from the learned corpus with about 65% characterized by 0.5 or less in textual distance. On the other hand, a word's neighborhoods that form automatically generated document usually stand further away with about 80% of them have a relatively higher distance. In short, out of the three methods to calculate similarity/distance that have been presented, textual distance seems to be able to provide the most distinction between types of document. textual distance for a set of F_A to learned corpus:

$Textualdist_{(A_{(w)},C_{(w)})} = \frac{1}{2} \times \sum_{x \in N_{A(w)} \cup N_{C(w)}} (N_{A(w)}x - N_{C(w)}x)$

where $N_{A(w)}(x) = \begin{cases} f(x \in N_{A(w)}), & \text{if } x \in N_{A(w)} \\ 0, & \text{otherwise} \end{cases}$

So, to conclude, this method would work best with textual distance on a word's neighborhood representation of the 20 most popular co-occurrence neighbors inside a window of five words. The next section will validate these proposals where each document is classified as generated or not

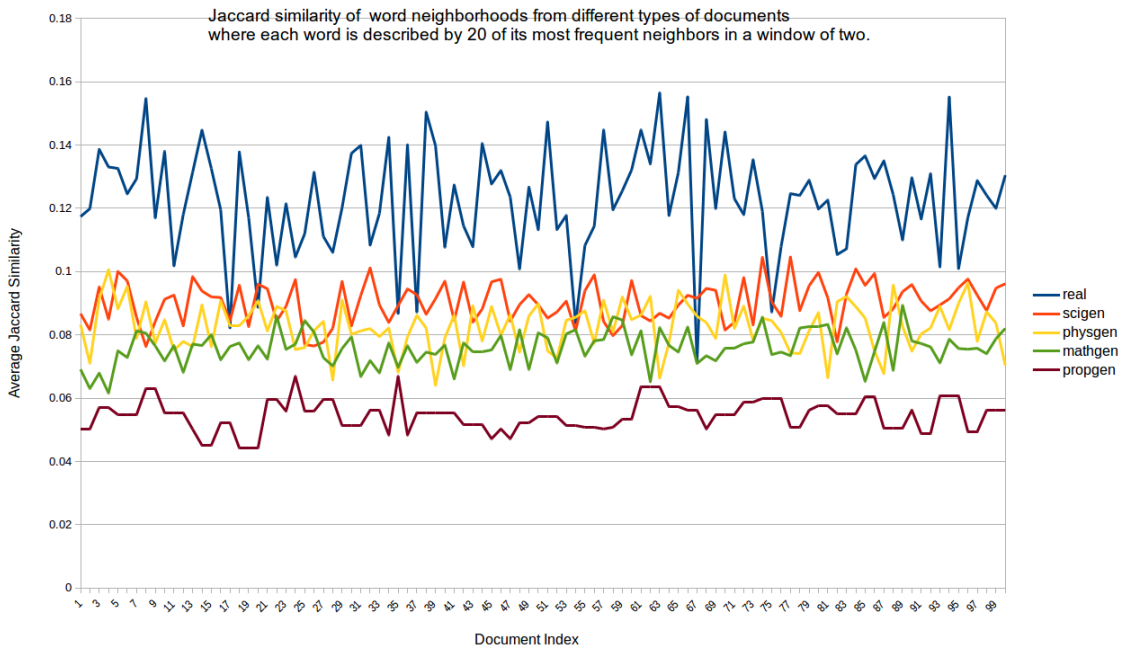


Figure 5.14: Jaccard similarity of frequent words neighborhood from different types of documents where each word is described by 20 of its most frequent neighbors in a window of two.

based on the percentage of the frequent keywords that have a "strange" neighborhood, strange progression of vocabulary or non-similar keywords from a small window.

5.4 Validation

This section aims to validate all the previously proposed hypotheses along with SciDetect as a method to detect documents with unusual word's distribution. Each method for detecting generated text without samples is set with thresholds as follows:

SciDetect: The same documents that were used to compose the LNSC corpus are used as references. If a document under test has higher than 0.6 in textual distance to its nearest neighbor inside the LNSC corpus, then it is considered as a non-genuine one.

Vocabulary growth: As presented in the corresponding Section 5.1.3, this method uses a combination of multiple vocabulary characteristics.

W2V similarity: Cosine similarity based on word vectors learned with the CBOW model is used and derived from Figure 5.7. If a document has more than half of its windows with higher than 0.19 in average pairwise cosine similarity, then it would be considered as not genuinely written.

Word neighborhood similarity: if more than half of the keywords have textual distance higher than 0.3 to the learned neighborhood then it would be classified as non-genuinely written.

These methods are tested with the Test corpus in Section 4.2.2 along with Markov and RNN corpora in Section 3.5 to fully understand what are the limitations of these proposals. The results

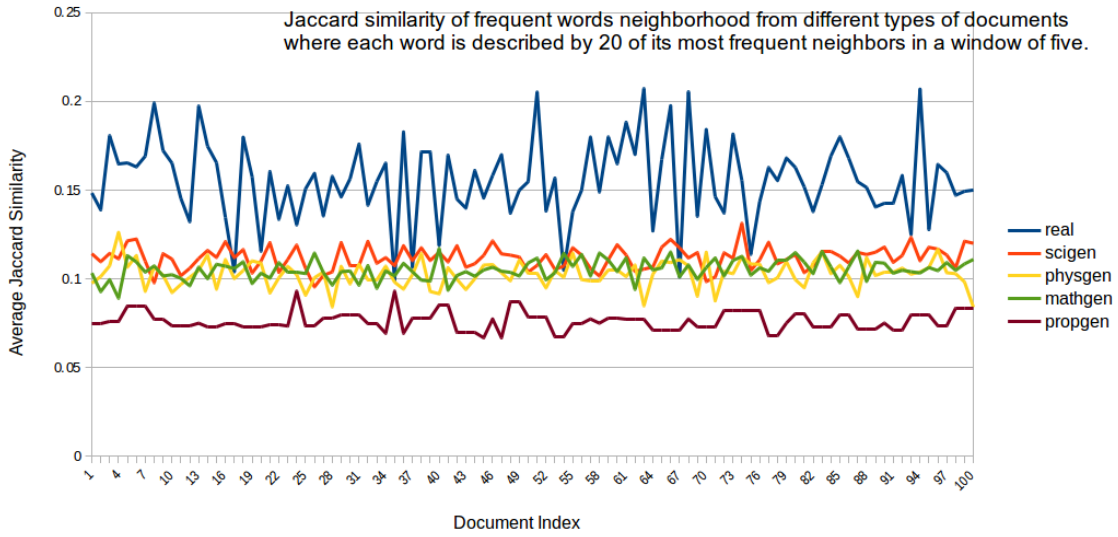


Figure 5.15: Jaccard similarity of frequent words neighborhood from different types of documents where each word is described by 20 of its most frequent neighbors in a window of five.

are shown in Table 5.2 where a false positive rate is the probability of a genuinely written document is classified as non genuine and false negative rate is when an automatically generated document is classified as genuinely written. The smaller these rates are, the better the result is.

	False Positive Rate	False Negative Rate		
	Genuinely written	PCFG Generated	Markov Model	RNN
SciDetect	0.35	0.65	0.45	0.27
Vocabulary Grown	0.17	0.20	0.68	0.15
W2V Similarity	0.34	0.65	0.8 0	0.77
Word's Neighborhood	0.13	0.27	0.31	0.10

Table 5.2: False positive and false negative rates of different methods on different corpora.

As in Table 5.2, SciDetect works quite poorly when using only genuinely written documents as references. About a third of the genuinely written documents in the test corpus was classified as automatically generated while a large number of automatically generated ones were labeled as genuinely written. The best result for SciDetect was when dealing with texts generated by a recurrent neural network. This is understandable since there are a number of misspelled words in those texts and that makes them share less common vocabulary with the learned corpus. Vocabulary growth, on the other hand, deals relatively well with the test corpus and RNN corpus since it was developed with that exact aim. However, with the case of the Markov model generation, this approach has less accuracy as the model could replicate the source text in word's distribution. Using average pairwise cosine similarity with representation vectors form Word2Vec seems to be the worst approach yet, even though it could correctly mark two thirds of genuinely written documents, a

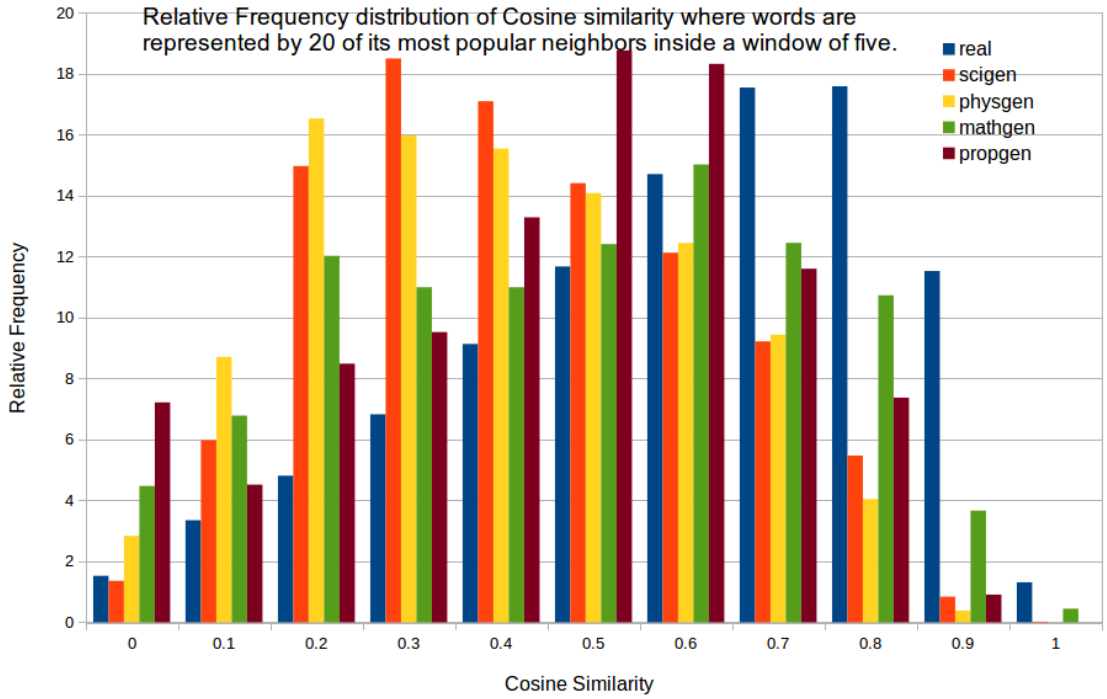


Figure 5.16: Relative Frequency distribution of cosine similarity for frequent words in the test corpus compared to the learned neighborhood where each word is represented by 20 of its most popular neighbors inside a window of five.

significant number of automatically generated documents were mistakenly marked as genuine. This means that, in general, keywords from generated documents are still some-what related to each other. Using a word's neighborhood appears to give a good result. Only 10% of genuinely written documents are mistakenly classified while 70% to 90% of automatically generated documents are spotted without any prior corpus sample about them.

5.5 Summary

This chapter has shown that it is difficult to positively pinpoint automatically generated sentences or documents using only genuinely written documents as references. This is understandable because of the richness of languages. There is no static model that can fully describe natural language even in a smaller area such as scientific documents. We have proposed several approaches for this problem from simply counting word types in a specific window or calculating the similarity of keywords inside a section of text. Furthermore, we developed a method to classify texts based on word's neighborhoods. These methods are then tested using a test corpus that includes both genuinely written documents and automatically generated documents from different generators, in addition to texts that were generated using a Markov model or a recurrent neural network. The results of the test confirm that it is a challenging task to correctly identify unknown automatically

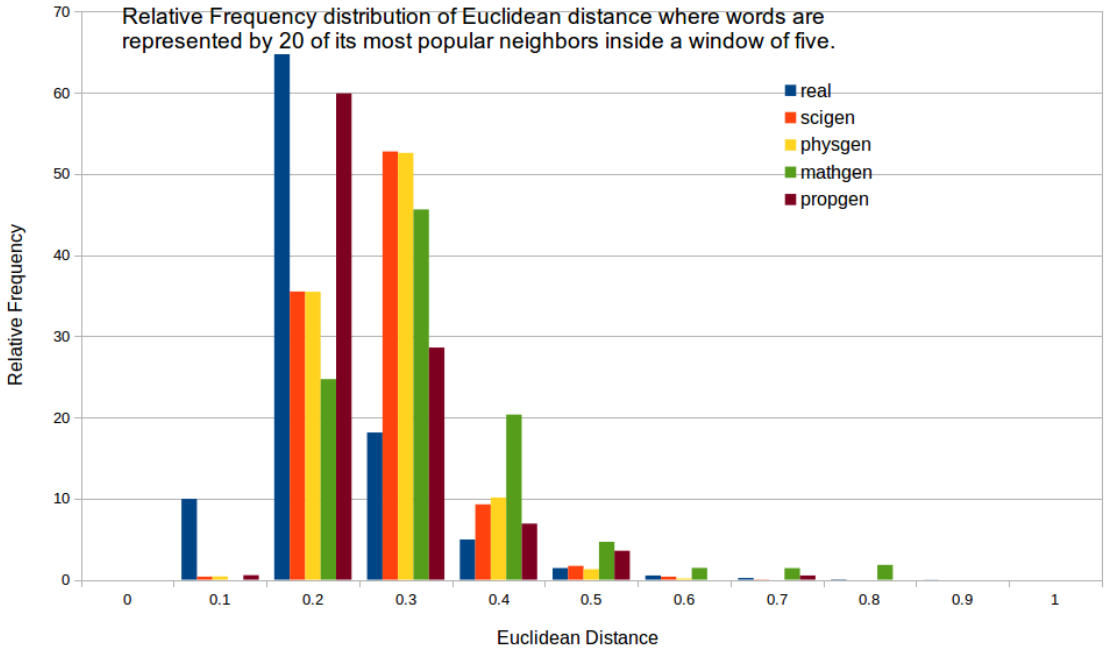


Figure 5.17: Relative Frequency distribution of Euclidean distance for frequent words in the test corpus compared to the learned neighborhood where each word is represented by 20 of its most popular neighbors inside a window of five.

generated documents. However, using word's neighborhoods along with textual distance, we were able to obtain a reasonably good result with a limited number of false positives which is the most important factor since, in reality, most if not all of the test documents should be genuinely written.

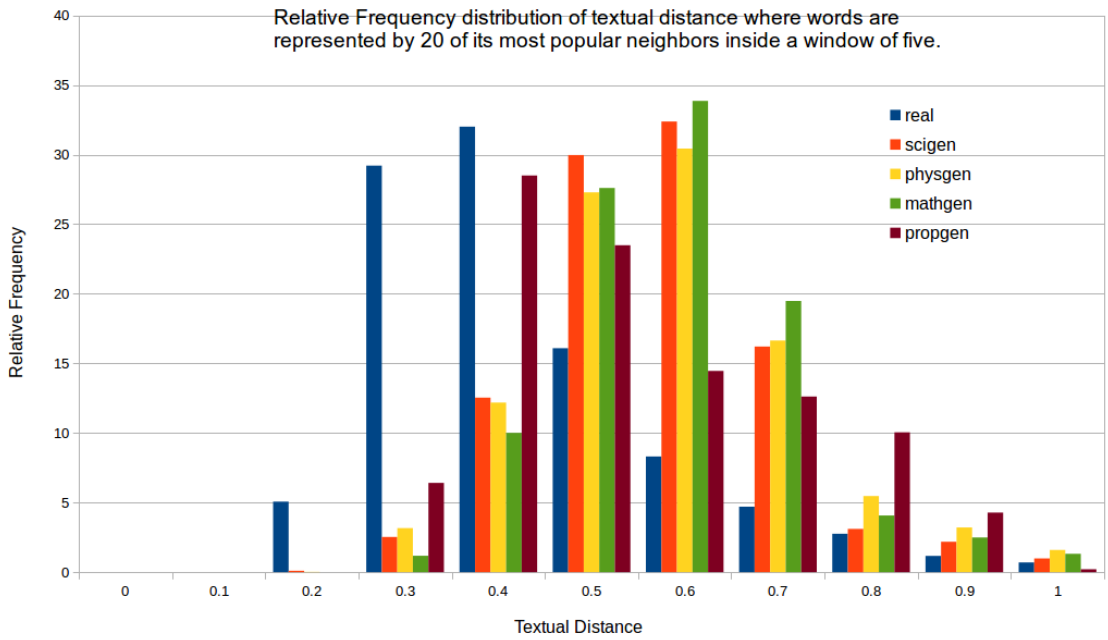


Figure 5.18: Relative Frequency distribution of textual distance for frequent words in the test corpus compared to the learned neighborhood where each word is represented by 20 of its most popular neighbors inside a window of five.

Chapter 6

SciDetect in an Industrial Environment

This chapter aims to give a broad view of a typical industrial publishing environment and how SciDetect has been integrated into a production workflow. This research is funded and realized in collaboration with Springer-Nature, one of the world's leading global research, educational and professional publishers. We were therefore able to collect valuable information to design a SciDetect system and to design the best integration into the workflow.

Springer-Nature is the world's largest academic book publisher, publisher of the world's most influential journals and a pioneer in the field of open research. The company numbers almost 13,000 staff in over 50 countries. Springer-Nature was formed in 2015 through the merger of Nature Publishing Group, Palgrave Macmillan, Macmillan Education and Springer Science+Business Media.

In 2016, there were more than five million articles available on Springer Link (the group's digital library) with approximate 265,000 new articles per year in 13,500 different issues, which means there are about 22,000 new articles per month. This is a real challenge to install a new system to the workflow which able to handle such a large volume without heavily impacting the existing systems.

6.1 The Publisher Workflow Experience

It is important for any publishing house to have a well-defined workflow. Furthermore, this workflow must also be highly standardized and well automated, but at the same time, stable and robust. This means the validity of each document must be ensured, including type-set, figures, tables, etc. Additionally, this workflow must provide a consistent environment for approximately 150 production editors all around the word.

This workflow is currently based on an XML structure, which is the basis for all products and deliverables (HTML, online and print PDF, ePub, ONIX, ...). This ensures standardized processing instructions for both vendors and printers with common interfaces to all vendors and other applications. These XML documents are stored in a centralized database called Data Delivery System (DDS) as the base for any kind of report and analysis. This database is maintained by a Process and Content Management department that directly works in tandem with us to implement SciDetect into the workflow. The overall process of the Process and Content Management systems

is shown in Figure 6.1 and each step will be briefly explained later.

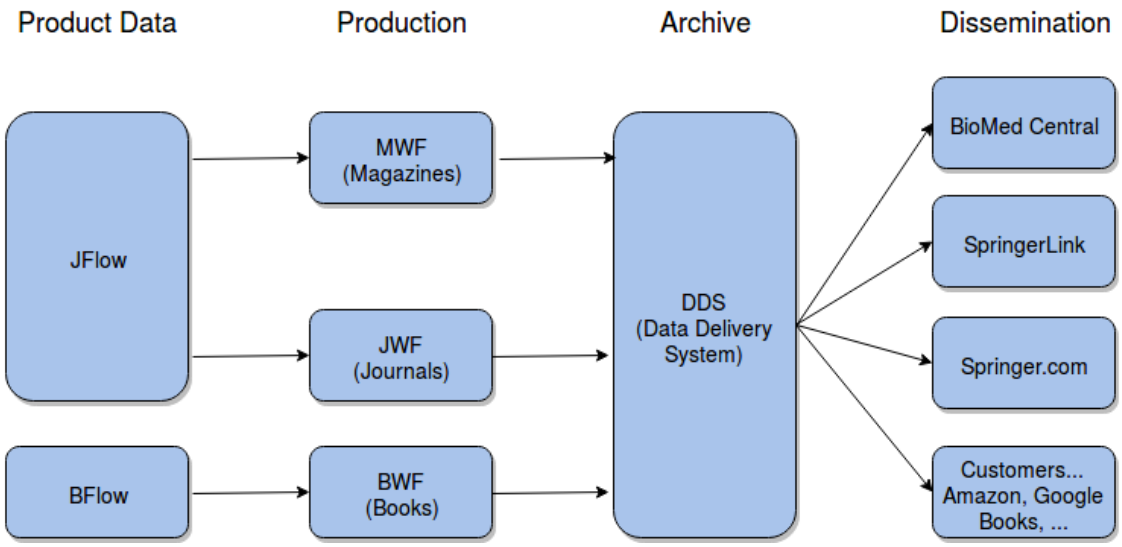


Figure 6.1: Process and Content Management Systems.

JFlow/BFlow: They are systems to store product's (article/book/journal) related information, as well as applications for maintaining current product metadata or planning tools for future information. In short, JFlow and BFlow are central repositories of journal and book production data. They are used mostly by editors for marketing and sales activities, along with report and analyses.

Journal workflow and Book workflow (JWF/BWF): These are the main pipeline to produce new articles or books. Figure 6.2 details these processes

- PRS (Peer Review System): Manuscript content is delivered to the system by automatically imported from an external Peer Review System.
- Initiate: The production editors perform initial tasks such as technical parameters for the article and bibliographic information.
- Author Query: The authors are contacted to transfer copyright and additional requests such as print in color, offprint.
- Prepare Content: The article is processed at a vendor to render as XML, images and separate PDF. A vendor in short is a service provider who digitizes the article in a specific format. This includes typesetting, correct formatting, exporting the article or book as a complete PDF, etc.
- Perform Proof: The delivery from the vendor is checked for quality by in-house production editors to ensure the formatting by a checker (submission checker). This step checks for various conditions such as XML formatting, reference to picture or LaTeX formula, as well as redacted content. All these small checks are controlled by a supervised system which is ready to report any detected problem to the appropriate manager. At the same time, corrections from authors are incorporated into the XML content of the article and its PDF rendering.

– Distributed Digital Content: After the quality check, the article is sent to the archiving system and delivered to SpringerLink, users, and other customers. Reaching this stage means the article is officially published with a registered DOI.

– DSS (Data Delivery System): The central database where the article is stored and distributed to different platforms.

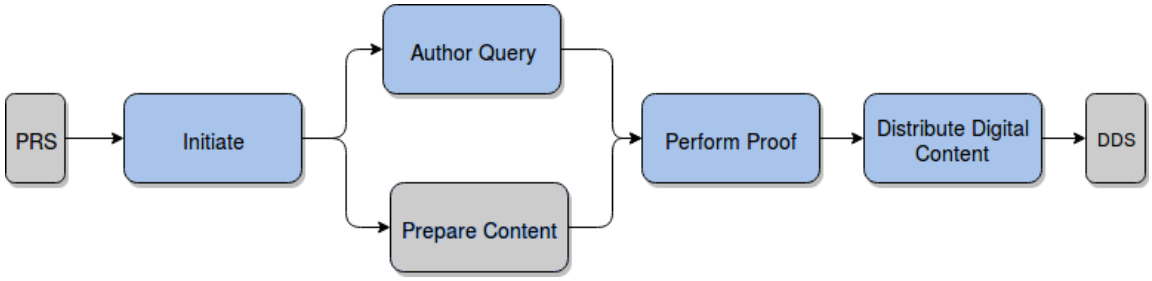


Figure 6.2: Article processing in JWF and BWF.

6.2 Incorporating SciDetect to the Workflow

Based on the previous section 6.1, it is desirable to have a semi-automatic system to process every new article that comes into the workflow. Thus, SciDetect is implemented as a Java program that runs by commands from the console and would also report via the console along with some log files for further inspection. The console report can be easily parsed and handled by another supervising system to notify an editor if there is a confirmed or suspected case of being automatically generated.

Furthermore, consistent results for each article must be kept and the ease of improvement for a new type of generator must also considered. Hence, SciDetect has been split and the computational part is deployed as a service running on a Tomcat server to provide connections to multiple instances of clients running on different workflows whenever needed. This ensures the availability of the system, as well as making any adjustment can be easily implemented without disturbing the workflow.

It is understandable to want to deploy SciDetect as early as possible. From Figure 6.2, the most desirable step to install SciDetect is between PRS and Initiate since the peer review system is not controlled by the internal workflow. However, at this step, there is no standard format (the article can come in Word, PDF or even LaTeX code) and there is no centralized observer system, so it would be costly and work intensive to develop a check at this position. Per contra, at the Perform Proof step, there is already a system in place to supervise different checks that need to be performed at this stage. Moreover, the input of this stage is a well-defined XML or PDF with a clean structure. This would make installing a SciDetect check at the Perform proof step as an extra required proof for the delivery an ideal situation.

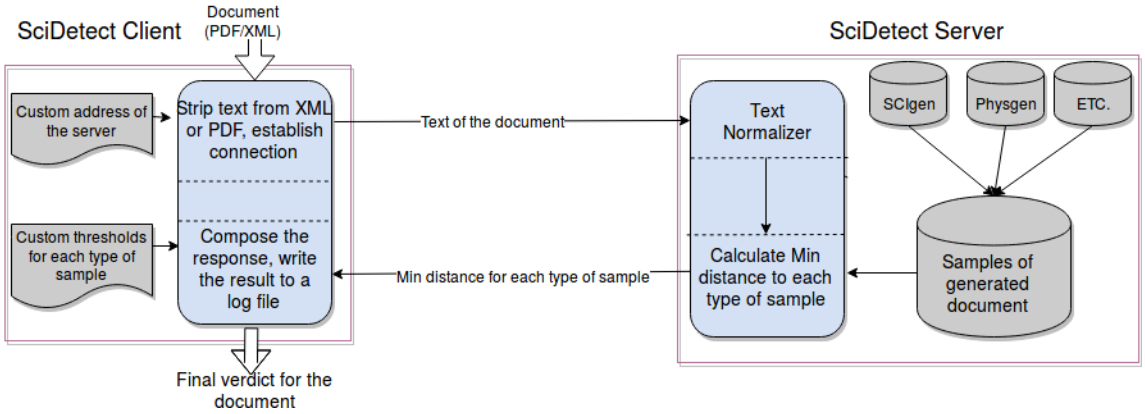


Figure 6.3: Architecture of SciDetect in client and server mode

In the end, the client's versions of SciDetect are installed in the stage Perform Proof of BWorkFlow and JWorkFlow, the overall architecture can be found in Figure 6.3. The SciDetect client is triggered whenever there is a new delivery from the vendors. The client first logs the time of calling along with the file name. Since each article can be delivered as a XML, a PDF or both, the SciDetect client prefers XML but if there is no XML available, the PDF version is targeted, and the plain text is striped from the document.

At the same time, a custom address of the server is obtained from a pre-defined configuration file and the connection between the client and the server is established. Using this connection, the text is then transferred to a SciDetect service running on a remote server where the text is normalized as presented in Chapter 3. The server also includes multiple corpora of samples from known generators and they are used to calculate the minimum distance to each type of sample corpus (as per algorithm in Chapter 3). The server side then responds to the client with the information about the file in relation to each of the sample corpus in the format: File name - Min distance to sample corpus - Sample corpus file name, as in Example 11.

Example 11 An example of a return result form SciDetect server for a file named "ICAART_2014_73.pdf ":

ICAART_2014_73.pdf	0.4840167178109426	data/samples/Scigen/INDEX-scigen25.txt
ICAART_2014_73.pdf	0.5121158731519747	data/samples/Physgen/INDEX-physgen31.txt
ICAART_2014_73.pdf	0.6345718723681273	data/samples/Mathgen/INDEX-mathgen95.txt
ICAART_2014_73.pdf	0.7459138216931677	data/samples/Propgen/INDEX-propgen11.txt

The client receives the distances and in turn uses its own configuration file to obtain the thresholds for each type of corpus and decides if the document can be considered as a generated one or not (Example 12). If it is considered as automatically generated or suspected generated, an error message is sent to the supervising system and it will trigger an automatic email to a responsible person to perform further testing. In any case, a log file is created that includes the time of the scan, the name of the scanned documents and what the results are, and this file can be used for further debugging if there is a problem.

Example 12 An example of final returns form SciDetect client

ICAART_2014_73.pdf	Suspected Scigen	0.4840167178109426
05359718.pdf	Mathgen	0.3511158731519747
456366.xtx	Genuine	0.6418756885197779

6.3 Preliminary Statistic Information and Interesting Lessons

SciDetect has been implemented and is working inside the workflows of Springer-Nature for about two years. And from the beginning of December 2016 to the end of November 2017, more than 600,000 articles have been processed and out of those numbers, 252 articles were marked as suspicious content. However, with closer inspection of those suspicious contents, all of them are genuinely written, but the minimum distance to the sample corpus is really close to the suspicious threshold. This suggests that the threshold can be fine-tuned further to avoid wasting resources on false positive cases.

Even though there are three main workflows for publishing, SciDetect was only deployed on two workflows for books and journals, so the workflow for magazine is unmodified. This is because the nature of this workflow is different; it only deals with articles from well-known, established editors and should be impervious to automatically generated documents.

We have also learned that to be able to handle a high number of production data, careful engineering is needed. During the earlier days of the implementation, there was several "road bumps." For example, we ran into a case of memory leak, and this problem was not revealed during the testing phrase in a small-scale simulation environment. However, when the service was deployed for a long continuous time, the leak kept compounding up, and the process required more and more RAM and eventually caused an out of memory error.

Another lesson is that detailed documentation needs to be implemented and maintained. An example of the documentation for different versions of SciDetect can be found in the Appendix. And continuous improvements need to be realized to keep up with requirements as an example of a change log for a new version as in Example 13.

Example 13 Example of a change log.

Version 2.4.5

- Added a new optional parameter to choose which type of file will be scanned: -type which can be either xml, pdf or both (defaulted to both).
  - If type "xml" is chosen and SciDetect encounter a A++ metadata only (does not contain a <Body> or a <PDFTextExtract>). SciDetect will try to locate the counterpart pdf at the appropriate location.
  - If "both" is chosen, there will be cases where SciDetect scans a pdf file twice (once when encounter the A++ meta data and once again for the pdf itself). Suggested usage: java -jar SciDetect_Local.jar -c pathtocheck -type xml
- Log file improvements.
  - Name format changed from YYYY.MM.DD-HH:mm.tsv to YYYY.MM.DDTHH:mm.tsv
  - Added a header line at the beginning of the log file to make it easier to be interpreted by a human reader when it is opened by a spread sheet program.
- Further define "cannot classify" error.

- If the file in question contains too little text (less than 10000 chars) to be confidently classified.
  - If it is a matter (book front, back matter or chapter matter) then SciDetect will say "Cannot Classify: File is a front or back matter".
  - Else "Cannot Classify: File does not contain enough text".
- If there was an error while converting the pdf, "Cannot Classify: An error occurring while converting file".

6.4 Summary

This chapter presented a brief overview of a real-life industrial workflow for a well-known publisher. Different stages of the workflow were introduced to find the best position to install a SciDetect check for an automatically generated article.

Moreover, since the installation of the system to the workflow, more than half a million articles have been processed, with a negligible number of false positive cases. Along with that, valuable lessons are learned during the development process about both engineering and providing continuous customer support.

Chapter 7

Conclusion

Automatically generated text has been used in numerous occasions with distinct intentions. This can simply go from generated comments in an online discussion to a much more mischievous task such as manipulating bibliographic information. Different methods to generate texts have been presented in Chapter 2, each with its own advantages, as well as disadvantages.

Then the thesis tackled multiple questions concerning the possibility of detecting automatically generated texts in various situations. Chapter 3 discussed current detection attempts to classify fully automatically generated documents (mostly SCIgen). Then we argued that it is possible to obtain good or better results by simply using the word distribution of the document to find out if it has a similar distribution to a known automatically generated document.

To validate our argument in this chapter, different distance/similarity measurements were tested to demonstrate that textual distance can provide the best separation. From that, the SciDetect system to classify automatically generated documents was built and tested against other known methods. Furthermore, the system was also proven to be able to detect documents that were generated by a Markov model or a recurrent neural network given that somehow samples from the same generator could be obtained.

Then Chapter 4 went one step further when proposing a new method to detect sentences or short paragraphs of automatically generated texts. As far as we know, this is the first approach trying to accomplish such a task. And to do it, we proposed a new similarity measurement called grammatical structure similarity. This similarity is calculated by comparing the parsed structure between two sentences.

The method uses a corpus of known sentences from different generators along with their parse tree. For each sentence under-test, itself along with its direct previous and next neighbors are compared to the corpus of known generated sentences to find similarities on both the lemma level and on the grammatical structure level. This method was then tested against some well-known machine learning techniques to demonstrate that it can provide better results. Furthermore, it is also proven to be able to detect sentences from a modified generator where only the terminal keywords are changed. However, if faced with texts generated by a Markov model or a recurrent neural network, this method is impractical since there is no re-occurrences in grammatical structures.

Chapter 5 took another direction. In this chapter, the hypothesis is that we are faced with a new automatic generator for which it is impossible to accumulate a substantial number of examples. This would mean that we can-not use a reference corpus of known generated samples as earlier chapters

did. To resolve this problem, we proposed to compare the document under test using different characteristics of genuinely written text.

First, the growth rate of the vocabulary and the diversity of the lemma is considered to show that automatically generated documents have distinct differences from genuinely written ones. Second, on another note, we tried to compare the keywords in the document under test to each other with the hypothesis that frequent keywords in a document must be somewhat related to each other. This was accomplished by using Word2Vec to build a vector to represent each word, and then those vectors are compared to each other with different methods.

Third, we assumed that the "normal" neighborhood of words can be used to differentiate PCFG automatically generated text since the terminal keywords are chosen at random, thus would result in an abnormal neighborhood. These methods were then tested on a test corpus that included both genuinely written documents and automatically generated documents from different generators, in addition to texts that were generated using a Markov model or a recurrent neural network. The results of these tests confirmed that it is a challenging task to correctly identify unknown automatically generated documents. However, using the word's neighborhood along with textual distance, we were able to obtain a reasonably good result with a limited number of false positives which is the most important factor, since in reality, most if not all the test documents should be genuinely written.

Finally, Chapter 6 presented an industrial environment in a high-profile publisher. From this information, detailed analysis was carried out to discover the best place as well as method to implement SciDetect into a real-life workflow. Some examples of the result for SciDetect inside the workflow were introduced to demonstrate the working process. Furthermore, important development lessons were learned during the development process and discussed.

However, the richness of human language always poses a challenge to confidently detect new types of generated documents. This calls for more in-depth research to be able to reach the very high reliability that is needed. This can be achieved by multiple methods such as trying to further model the language based on its characteristics or by employing a name entity recognition [Quimbaya et al., 2016]. This information can be used not only to detect automatically generated documents but also to classify genuinely written documents into appropriate sub-classes. Furthermore, it might be able to adapt to be used as a fake news detector which is a very active field of research at the moment [Conroy et al., 2015, Sethi, 2017, Shu et al., 2017, Volkova et al., 2017].

Another interesting research direction that can be explored is about error detection in scientific documents. This might be done by using the embedded vectors to discover unusual combinations of words or as [Byrne and Labbé, 2017] by using a more complicated workflow. This problem has also been highlighted on multiple occasions [Phillips, 2017b, Phillips, 2017a]

In conclusion, this thesis has shed light on multiple important research questions about the possibility of detecting automatically generated texts in different settings. Beside the research aspect, important engineering work in a real-life industrial environment is also carried out to demonstrate that it is important to have real application along with fundamental research. The results from this thesis are curently in production at the world leading academic publisher. And it can be expanded and used in multiple future problems such as improving the review process [Hartley and Cabanac, 2017] or authorship attribution [Savoy, 2012]. Furthermore, this field of research is still fertile and can still be explored much deeper.

Bibliography

[Altman, 1992] Altman, N. S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185.

[Amancio, 2015] Amancio, D. R. (2015). Comparing the topological properties of real and artificially generated scientific manuscripts. *Scientometrics*, 105(3):1763–1779.

[Ball, 2005] Ball, P. (2005). Computer conference welcomes gobbledegook paper. *Nature*, 434, 946.

[Barbieri et al., 2012] Barbieri, G., Pachet, F., Roy, P., and Esposti, M. D. (2012). Markov constraints for generating lyrics with style. In *Proceedings of the 20th European Conference on Artificial Intelligence*, pages 115–120. IOS Press.

[Beel and Gipp, 2010] Beel, J. and Gipp, B. (2010). Academic search engine spam and Google scholar's resilience against it. *Journal of Electronic Publishing*.

[Beel et al., 2010] Beel, J., Gipp, B., and Wilde, E. (2010). Academic search engine optimization (ASEO). *Journal of scholarly publishing*, 41(2):176–190.

[Bender and Wolf, 1991] Bender, P. E. and Wolf, J. K. (1991). New asymptotic bounds and improvements on the Lempel-Ziv data compression algorithm. *IEEE Transactions on Information Theory*, 37(3):721–729.

[Bohannon, 2013] Bohannon, J. (2013). Who's afraid of peer review? *Science*, 342(6154):60–5.

[Broder et al., 1997] Broder, A. Z., Glassman, S. C., Manasse, M. S., and Zweig, G. (1997). Syntactic clustering of the web. *Comput. Netw. ISDN Syst.*, 29(8-13):1157–1166.

[Bulhak, 1996] Bulhak, A. (1996). On the simulation of postmodernism and mental debility using recursive transition networks. Technical report, Departement of Computer Science, Monash University.

[Byrne and Labbé, 2017] Byrne, J. A. and Labbé, C. (2017). Striking similarities between publications from China describing single gene knockdown experiments in human cancer cell lines. *Scientometrics*, 110(3):1471–1493.

[Cer et al., 2010] Cer, D. M., De Marneffe, M.-C., Jurafsky, D., and Manning, C. D. (2010). Parsing to Stanford dependencies: Trade-offs between speed and accuracy. In *LREC*. Floriana, Malta.

[Chomsky, 1956] Chomsky, N. (1956). Three models for the description of language. *IEEE Transactions on Information Theory*, 2(2):113–124.

[Chomsky, 1959] Chomsky, N. (1959). On certain formal properties of grammars. *Information and control*, 2(2):137–167.

[Chomsky, 2002] Chomsky, N. (2002). *Syntactic structures*. Walter de Gruyter.

[Collingwood et al., 2013] Collingwood, L., Jurka, T., Boydstun, A., Grossman, E., and van Atteveldt, W. (2013). Rtexttools: A supervised learning package for text classification. *The R Journal*, 5(1):6–13.

[Conroy et al., 2015] Conroy, N. J., Rubin, V. L., and Chen, Y. (2015). Automatic deception detection: Methods for finding fake news. *Proceedings of the Association for Information Science and Technology*, 52(1):1–4.

[Culotta and Sorensen, 2004] Culotta, A. and Sorensen, J. (2004). Dependency tree kernels for relation extraction. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, ACL '04, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Dalkilic et al., 2006] Dalkilic, M. M., Clark, W. T., Costello, J. C., and Radivojac, P. (2006). Using compression to identify classes of inauthentic texts. In *Proc. of the 2006 SIAM Conf. on Data Mining*.

[Delgado López-Cózar et al., 2014] Delgado López-Cózar, E., Robinson-García, N., and Torres-Salinas, D. (2014). The Google scholar experiment: How to index false papers and manipulate bibliometric indicators. *Journal of the Association for Information Science and Technology*, 65(3):446–454.

[Durán et al., 2014] Durán, K., Rodríguez, J., and Bravo, M. (2014). Similarity of sentences through comparison of syntactic trees with pairs of similar words. In *Electrical Engineering, Computing Science and Automatic Control (CCE)*, pages 1–6.

[Fahrenberg et al., 2014] Fahrenberg, U., Biondi, F., Corre, K., Jegourel, C., Kongshøj, S., and Legay, A. (2014). Measuring global similarity between texts. In Besacier, L., Dediu, A.-H., and Martín-Vide, C., editors, *Statistical Language and Speech Processing*, pages 220–232, Cham. Springer International Publishing.

[Feinerer et al., 2008] Feinerer, I., Hornik, K., and Meyer, D. (2008). Text mining infrastructure in R. *Journal of Statistical Software*, 25(5):1–54.

[Firth, 1957] Firth, J. R. (1957). A synopsis of linguistic theory, 1930-1955. *Studies in linguistic analysis*.

[Friedman et al., 2010] Friedman, J., Hastie, T., and Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22.

[Friedman et al., 2000] Friedman, J., Hastie, T., Tibshirani, R., et al. (2000). Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407.

[Ganguly et al., 2015] Ganguly, D., Roy, D., Mitra, M., and Jones, G. J. (2015). Word embedding based generalized language model for information retrieval. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 795–798. ACM.

[Ginsparg, 2014] Ginsparg, P. (2014). Automated screening: ArXiv screens spot fake papers. Nature - 508:44.

[Graves, 2013] Graves, A. (2013). Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850.

[Hartley and Cabanac, 2017] Hartley, J. and Cabanac, G. (2017). What can new technology tell us about the reviewing process for journal submissions in *BJET*? *BJET*, 48(1):212–220.

[Herrera et al., 2006] Herrera, J., Peñas, A., Rodrigo, Á., Verdejo, F., Magnini, B., and Dagan, I. (2006). UNED at PASCAL RTE-2 challenge. *2nd PASCAL Challenges Workshop on Recognising Textual Entailment*, pages 38–43.

[Huttenlocher et al., 1991] Huttenlocher, J., Haight, W., Bryk, A., Seltzer, M., and Lyons, T. (1991). Early vocabulary growth: Relation to language input and gender. *Developmental psychology*, 27(2):236.

[Kao, 2017] Kao, J. (accessed Nov 2017). More than a million pro-repeal net neutrality comments were likely faked - https://hackernoon.com/more-than-a-million-pro-repeal-net-neutrality-comments-were-likely-faked-e9f0e3ed36a6.

[Karpathy, 2017] Karpathy, A. (accessed Nov 2017). The unreasonable effectiveness of recurrent neural networks - http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

[Klein and Manning, 2003] Klein, D. and Manning, C. D. (2003). Fast exact inference with a factored model for natural language parsing. pages 3–10.

[Kullback, 1959] Kullback, S. (1959). *Information Theory and Statistics*. Wiley, New York.

[Kullback and Leibler, 1951] Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86.

[Labbé, 2010] Labbé, C. (2010). Ike Antkare one of the great stars in the scientific firmament. *ISSI Newsletter*, 6(2):48–52.

[Labbé and Labbé, 2006] Labbé, C. and Labbé, D. (2006). A tool for literary studies: Intertextual distance and tree classification. *LLC*, 21(3):311–326.

[Labbé and Labbé, 2012] Labbé, C. and Labbé, D. (2012). Detection of hidden intertextuality in the scientic publications. In *International Conference on Textual Data Statistical Analysis. JADT 2012*.

[Labbé and Labbé, 2013] Labbé, C. and Labbé, D. (2013). Duplicate and fake publications in the scientific literature: How many SCIgen papers in computer science? *Scientometrics*, 94(1):379–396.

[Labbé and Labbé, 2014] Labbé, C. and Labbé, D. (2014). Was Shakespeare's vocabulary the richest? In *Proceedings of the 12th International Conference on Textual Data Statistical Analysis*, pages 323–336, Paris.

[Labbé et al., 2015] Labbé, C., Labbé, D., and Portet, F. (2015). *Creativity and Universality in Language*, chapter Detection of computer generated papers in scientific literature. Lecture Notes in Morphogenesis. Lecture Notes in Morphogenesis, Springer-Verlag.

[Labbé et al., 2016] Labbé, C., Labbé, D., and Portet, F. (2016). *Detection of Computer-Generated Papers in Scientific Literature*, pages 123–141. Springer International Publishing.

[Lavoie and Krishnamoorthy, 2010] Lavoie, A. and Krishnamoorthy, M. (2010). Algorithmic detection of computer generated text. *arXiv preprint arXiv:1008.0706*.

[Liaw and Wiener, 2002] Liaw, A. and Wiener, M. (2002). Classification and regression by random forest. *R News*, 2(3):18–22.

[López-Cózar et al., 2012] López-Cózar, E. D., Robinson-García, N., and Torres-Salinas, D. (2012). Manipulating Google Scholar citations and Google Scholar metrics: Simple, easy and tempting. *arXiv preprint arXiv:1212.0638*.

[Medelyan et al., 2009] Medelyan, O., Frank, E., and Witten, I. H. (2009). Human-competitive tagging using automatic keyphrase extraction. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3 - Volume 3*, EMNLP '09, pages 1318–1327, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Mikolov, 2012] Mikolov, T. (2012). *Statistical language models based on neural networks*. PhD thesis, Brno University of Technology.

[Mikolov et al., 2013a] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[Mikolov et al., 2011] Mikolov, T., Deoras, A., Povey, D., Burget, L., and Černockỳ, J. (2011). Strategies for training large scale neural network language models. In *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*, pages 196–201. IEEE.

[Mikolov et al., 2013b] Mikolov, T., Le, Q. V., and Sutskever, I. (2013b). Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*.

[Mikolov et al., 2013c] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013c). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

[Mikolov et al., 2013d] Mikolov, T., Yih, W.-t., and Zweig, G. (2013d). Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751.

[Myers et al., 1994] Myers, L. E., McQuay, L. J., and Hollinger, F. B. (1994). Dilution assay statistics. *Journal of clinical microbiology*, 32(3):732–739.

[Nguyen and Labbé, 2016] Nguyen, M. and Labbé, C. (2016). Engineering a tool to detect automatically generated papers. In *Proceedings of the Third Workshop on Bibliometric-enhanced Information Retrieval co-located with the 38th European Conference on Information Retrieval (ECIR 2016)*, pages 54–62.

[Palangi et al., 2016] Palangi, H., Deng, L., Shen, Y., Gao, J., He, X., Chen, J., Song, X., and Ward, R. (2016). Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 24(4):694–707.

[Pedersen et al., 2004] Pedersen, T., Patwardhan, S., and Michelizzi, J. (2004). Wordnet::similarity: Measuring the relatedness of concepts. In *Demonstration Papers at HLT-NAACL 2004*, HLT-NAACL–Demonstrations '04, pages 38–41, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1532–1543.

[Phillips, 2017a] Phillips, N. (accessed Nov 2017a). Natures 10 ten people who mattered this year -Jennifer Byrne: Error sleuth - https://www.nature.com/immersive/d41586-017-07763-y/index.html.

[Phillips, 2017b] Phillips, N. (accessed Nov 2017b). Online software spots genetic errors in cancer papers - https://www.nature.com/news/online-software-spots-genetic-errors-in-cancer-papers-1.23003.

[Quimbaya et al., 2016] Quimbaya, A. P., Múnera, A. S., Rivera, R. A. G., Rodríguez, J. C. D., Velandia, O. M. M., Peña, A. A. G., and Labbé, C. (2016). Named entity recognition over electronic health records through a combined dictionary-based approach. *Procedia Computer Science*, 100(Supplement C):55 – 61. International Conference on ENTERprise Information Systems/International Conference on Project MANagement/International Conference on Health and Social Care Information Systems and Technologies, CENTERIS/ProjMAN / HCist 2016.

[Savoy, 2012] Savoy, J. (2012). Authorship attribution based on specific vocabulary. *ACM Trans. Inf. Syst.*, 30(2):12:1–12:30.

[Sethi, 2017] Sethi, R. J. (2017). Crowdsourcing the verification of fake news and alternative facts. In *Proceedings of the 28th ACM Conference on Hypertext and Social Media*, HT '17, pages 315–316, New York, NY, USA. ACM.

[Shu et al., 2017] Shu, K., Sliva, A., Wang, S., Tang, J., and Liu, H. (2017). Fake news detection on social media: A data mining perspective. *SIGKDD Explor. Newsl.*, 19(1):22–36.

[Singhal, 2001] Singhal, A. (2001). Modern Information Retrieval: A Brief Overview. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 24(4):35–42.

[Sochenkov et al., 2016] Sochenkov, I., Zubarev, D., Tikhomirov, I., Smirnov, I., Shelmanov, A., Suvorov, R., and Osipov, G. (2016). Exactus like: Plagiarism detection in scientific texts. In *European Conference on Information Retrieval*, pages 837–840.

[Sokal, 1996] Sokal, A. D. (1996). Transgressing the boundaries: Toward a transformative hermeneutics of quantum gravity. *Social Text*, (46/47):217–252.

[Sutskever et al., 2011] Sutskever, I., Martens, J., and Hinton, G. E. (2011). Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024.

[Tang et al., 2014] Tang, D., Wei, F., Yang, N., Zhou, M., Liu, T., and Qin, B. (2014). Learning sentiment-specific word embedding for twitter sentiment classification. In *ACL (1)*, pages 1555–1565.

[Van-Noorden, 2014] Van-Noorden, R. (2014). Publishers withdraw more than 120 gibberish papers. *Nature News*.

[Volkova et al., 2017] Volkova, S., Shaffer, K., Jang, J. Y., and Hodas, N. (2017). Separating facts from fiction: Linguistic models to classify suspicious and trusted news posts on Twitter. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 647–653.

[Wang and Neumann, 2007] Wang, R. and Neumann, G. (2007). Recognizing textual entailment using sentence similarity based on dependency tree skeletons. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, RTE '07, pages 36–41, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Williams and Giles, 2015] Williams, K. and Giles, C. L. (2015). On the use of similarity search to detect fake scientific papers. In *Similarity Search and Applications - 8th International Conference, SISAP 2015*, pages 332–338.

[Xiong and Huang, 2009] Xiong, J. and Huang, T. (2009). An effective method to identify machine automatically generated paper. In *Knowledge Engineering and Software Engineering*, pages 101–102.

[Ziv and Lempel, 2006] Ziv, J. and Lempel, A. (2006). A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theor.*, 23(3):337–343.

[Zou et al., 2013] Zou, W. Y., Socher, R., Cer, D., and Manning, C. D. (2013). Bilingual word embeddings for phrase-based machine translation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1393–1398.

[Zubarev and Sochenkov, 2014] Zubarev, D. and Sochenkov, I. (2014). Using sentence similarity measure for plagiarism source retrieval. In *CLEF (Working Notes)*, pages 1027–1034.

[Zweig and Burges, 2011] Zweig, G. and Burges, C. J. (2011). The Microsoft Research sentence completion challenge. Technical report.

Appendices

Appendix A

Some More Examples of Known Partially Generated Papers

Use of Cloud-Computing and Social Media to Determine Box Office Performance

Riaa Seth[1], Baldev Singh[2]
[1]Student at Sokal Insititute of Technology, Pune, [2]Lecturer at Sokal Institute of Technology, Pune

***Abstract*— Determining the box-office performance of a movie remains one of the most challenging grand problems posed by Hilbert. It has not only theoretical implications, but the commercial value of a solution would also be tremendous. In this paper we show that this problem can be solved to a high degree of accuracy through the use of cloud computing techniques, and the use of social media networks. We also provide a hybrid approach which works the best.**

***Index Terms*—HTTP Protocol, Facebook, Twitter, Internet**

I. INTRODUCTION

Interrupts must work. Unfortunately, a typical quagmire in cyberinformatics is the investigation of "fuzzy" models. After years of essential research into web browsers, we argue the deployment of Boolean logic, which embodies the practical principles of e-voting technology. The improvement of reinforcement learning would tremendously amplify Bayesian models.

As is clear from the title of this paper, this paper deals with the entertainment industry. So, we do provide entertainment in this paper. So, if you are reading this paper for entertainment, we suggest a heuristic that will allow you to read this paper efficiently. You should read any paragraph that starts with the first 4 words in bold and italics – those have been written by the author in painstaking detail. However, if a paragraph does not start with bold and italics, feel free to skip it because it is gibberish auto-generated by the good folks at SCIGen.

II. Cloud-based UIB

Despite Xeno's paradox, we can demonstrate that use of cloud-computing technologies can be beneficial in this case[18] and can be made embedded, ubiquitous, and linear-time. Specifically we describe the UIB algorithm, whose architectural overview is given in Figure 1.

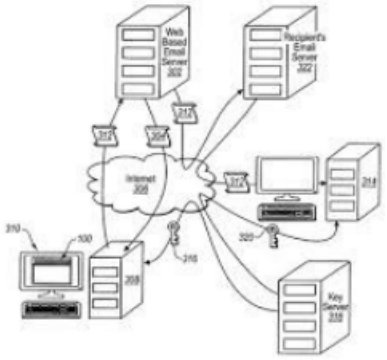


Figure 1: The architectural layout used by UIB

Typically, a data center is a facility within a computing environment that is used to house mission critical computer systems and associated components. A data center includes environmental controls, such as air conditioning, fire suppression, and the like, redundant/backup power supplies, redundant data communications connections and high security among

Figure A.1: An excerpt from a partially generated paper by Navin Kabra where a genuinely written paragraph is marked in blue.

refines the construction of location-aware pairs between mobile agents, but without all the unnecssary complexity. The choice of mobile agents in [16] differs from ours in that we synthesize our algorithms in hierarchical location-identity splits. Continuing with this rationale, the matching-based algorithms proposed by Johnson and Smith, however, fails to address several key issues that LISP does answer.

Several semantic and "fuzzy" heuristics have been proposed [10]. Without using random assumption for the newly added connection, it is not hard to imagine that the Bayesian algorithm for the location-aware communications is not recursively enumerable. Recently, a novel methodology for the investigation [12] of the emulation of the location-identity split addresses several key issues that our system also surmount.

Our heuristic builds on existing work in pervasive theory and algorithms. LISP is broadly related to work in the field of artificial intelligence by Duquennoy [8], but we view it from a new perspective: the deployment of the location-identity split [12]. As a result, the class of frameworks enabled by our framework is fundamentally different from related solutions [5]. We would like to answer the challenges inherent in the existing work, since the original solution to this obstacle did not completely answer this issue [16].

III. Methodology

A. Location-Identity SPlit (LISP)

An example of Location-Identity SPlit (LISP) is showed in Figure 1.

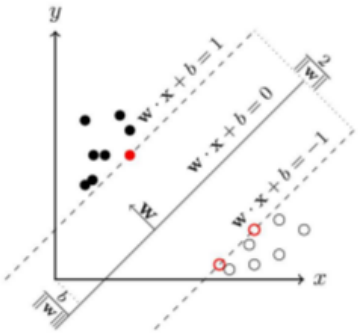


Fig. 1. One example of the Location-Identity SPlit (LISP). The diagonal solid black line is a valid split with a fixed width. Agents with red color are boundary agents, while agents with black color are mobile agents.

It would be interesting to apply the similar techniques [8] to show that the newly and randomly generated connections can be made concurrent, signed, and replicated. Assume every agent is uniquely defined,

Definition III.1. *Let $\lambda \geq 0$ be arbitrary. A connection between two agents a_1 and a_2 is λ-**concurrent** if the location-identity splits has at least λ configurations.*

Definition III.2. *Let $\gamma \geq 0$ be arbitrary. A connection matrix $g(\mathcal{X})$ is γ-**signed** if $g(\mathcal{X} + \gamma\mathcal{I})$ is isometric and separable.*

Definition III.3. *Let $d \geq 0$ be arbitrary. A connection between two agents a_1 and a_2 is d-**replicated** if there exists at least d splits that seperate a_1 and a_2.*

Lemma III.1. *Suppose*

a) *let $\gamma \geq 0$ be arbitrary, and $g(\mathcal{X})$ is γ-**signed**,*
b) *let $d \geq 0$ be arbitrary, and the connection between two agents a_1 and a_2 is d-**replicated**,*
c) *let $\lambda \geq 0$ be arbitrary, and the connection between two agents a_1 and a_2 is λ-**concurrent**.*

Show that

$$\int_{\gamma}^{\infty} g(\mathcal{X} + \gamma\mathcal{I})d\lambda \text{ is convergant} \iff \int_{\gamma}^{\infty} g(\mathcal{X})d\lambda \text{ is convergent.}$$

Proof: For $\lambda > \gamma$,

$$\int_{\gamma}^{\infty} g(\mathcal{X} + \gamma\mathcal{I})d\lambda = \int_{\gamma}^{\lambda} g(\mathcal{X} + \gamma\mathcal{I})d\mathcal{I} + \int_{\lambda}^{\infty} g(\mathcal{X})d\mathcal{X}.$$

By the definition of an indefinite integral,

$$\int_{\lambda}^{\infty} g(\mathcal{X})d\mathcal{X} = \lim_{n\to\infty} \int_{\lambda}^{n} g(\mathcal{X})d\mathcal{X}.$$

We can split this up into $\int_{\gamma}^{\lambda} g(\mathcal{X} + \gamma\mathcal{I})d\mathcal{I} + \int_{\lambda}^{\infty} g(\mathcal{X})d\mathcal{X}$ because we now have a definite integral. Again using the definition of an indefinite integral we have

$$\int_{\gamma}^{\lambda} g(\mathcal{X} + \gamma\mathcal{I})d\mathcal{I} + \int_{\lambda}^{\infty} g(\mathcal{X})d\mathcal{X} = \lim_{n\to\infty} \int_{\lambda}^{n} g(\mathcal{X})d\mathcal{X}.$$

Now, since we are trying to show an if and only if relationship, we must prove the implication both ways. First, let $\int_{\gamma}^{\infty} g(\mathcal{X} + \gamma\mathcal{I})d\lambda$ be convergant. We must show that $\int_{\gamma}^{\infty} g(\mathcal{X})d\lambda$ is convergant. We have that $\int_{\lambda}^{\infty} g(\mathcal{X})d\mathcal{X} = \int_{\gamma}^{\lambda} g(\mathcal{X} + \gamma\mathcal{I})d\mathcal{I} + \int_{\lambda}^{\infty} g(\mathcal{X})d\mathcal{X}$. It is given that $\int_{\gamma}^{\infty} g(\mathcal{X})d\lambda$ is convergant. $\int_{\lambda}^{\infty} g(\mathcal{X})d\mathcal{X}$ is simply a definite integral, so we know that it has some constant value. With these two

Figure A.2: An excerpt from a partially generated paper that was submitted to ICAART 2014 with an unknown number of generated sentences.

SciDetect™ Documentation

http://scidetect.forge.imag.fr

Nguyen Minh Tien
Minh-Tien.nguyen@imag.fr
Cyril Labbé
first.last@imag.fr

March 2015

Revision History

Version	Date	Author	Comment
1.4	13-02-2015	MT	Initial deployment
1.41	17-02-2015	MT	Added support for XML and XTX
2.0	25-02-2015	MT	Added multiple configurable parameters
2.1	09-03-2015	MT	Separated text and corpus class
2.2	25-03-2015	MT	Added Web Service options

Appendix B

SciDetect Local

This chapter is for the localized version of SciDetect where everything is performed on a local machine.

B.1 Installation-Requirements-Quick start

Installation A stand-alone Java program, the documentation and the source code are available at the following URL: `http://scidetect.forge.imag.fr`

Requirements The stand-alone Java program requires `Java SE 6` or higher. It also uses an additional library for PDF converter (should be included in the `lib/` directory).

Quick start The runnable program for the SciDetect software is packaged inside:

```
SciDetect_Local.jar
```

The following are needed :

- The configuration file (`config.txt`)
- The samples directory directories (`web/WEB-INF/data`)

B.2 Usage

B.2.1 Command line client

SciDetect program is included in a runnable JAR file. The program is started by invoking:

```
$java -jar SciDetect_Local.jar <parameters>
```

Where `<parameters>` stands for a combination of one or more of the following command line options:

`-c <path_to_check>` gives the path to the directory containing the files to be checked;

`-l <log_filename>` gives the name of the log file (defaults to `/logs/start_time.xls`);

`-d` Save detail log (optional, default false).

`-h` Show usage.

Typical use:

```
$java -jar SciDetect_Local.jar -c /tien/Test_demo -l /tien/Test_log.xls -d
```

B.2.2 Supported file types

At version 2.1 `SciDetect_Local` currently supports .PDF and two specific Springer xml format namely `.XML` for `A++` format `.XTX` for PDF extraction of PDF files

B.3 Configuration

A configuration file (`config.txt`) should be accessible by the program. It should be found in the same directory with the `SciDetect_Local.jar`. The config file should contain the following information:

B.3.1 Path to sample folder

```
# Where samples can be found
samples web/WEB-INF/data/samples
```

This is used to set the directory where samples of texts produced by known generators can be found. This directory contains one directory per *class* (i.e. per known generator). One directory contains examples that are representative of its class. In a standard release, the `web/WEB-INF/data/samples` directory contains four subdirectories with texts generated by the following generator:

- `http://thatsmathematics.com/mathgen/` (dir `data/samples/Mathgen`);
- `https://bitbucket.org/birkenfeld/scigen-physics` (dir `data/samples/Physgen`);
- `http://www.nadovich.com/chris/randprop/` (dir `data/samples/Propgen`);
- `http://pdos.csail.mit.edu/scigen/` (dir `data/samples/SCIgen`).

New subdirectories can be added. This can be done for two purposes:

1. Adding a corpus that represents fairly enough of a particular field. By setting an appropriate threshold, this will flag papers that appear to be too far from that field.
2. When a generator appears, new samples (pdf) can be added in a new subdirectory (in `data/samples`) containing representative corpora of the new class.

B.3.2 Threshold configuration

```
# Defining Thresholds for SCIgen
Threshold_Scigen        0.48     0.56
```

A line starting with `Threshold_Dirname` is used to define thresholds. Thresholds are needed to make decisions to assigned tested texts to a class. Examples of each class can be found in the directory `Dirname`. There should have one line (i.e. two Thresholds) per class. These values are 2 real numbers between 0 and 1. The smallest one is used to make the decision to assign the tested paper (almost certainly) to the class. The second one is used as a threshold for suspicion for containing parts of generated text.

The previous example (concerning SCIgen class) has the following meaning. Given distances from the tested text to its nearest neighbour in the set of samples (i.e. texts found in the Scigen dir):

- If the distance is greater than 0.56, then it is reasonably believable that this is a genuine article.
- From 0.56 to 0.48, there is a chance that this article or part of this article is Scigen generated.
- If the distance is less than 0.48, there is a very high chance that this is an automatic Scigen generated article.

If new samples are added to the sample folder (i.e. new dir), the threshold configuration should also be added, if not the default-threshold values are used (0.48 and 0.56).

B.3.3 Path for log files

```
# Set the default path for log files
Default_log_folder        logs/
Default_detail_log_folder        detaillogs/
```

These lines are used to set the default log folder and a default detail log folder (see section D.1 for more information). In case the path to a log file is not set (no -l parameter), the log file will be saved in the default log folder under the name: `time_date.xls` (e.g. 09:46 25.02.2015.xls means the check was started at 9:46 on 25/2/2015).

```
INDEX-53.txt    is a Scigen     0.34236384      data/samples/Scigen/INDEX-scigen25.txt
INDEX-53.txt    is a Physgen    0.47908222      data/samples/Physgen/INDEX-physgen7.txt
INDEX-011.txt   is Genuine      0.60918242      data/samples/Scigen/INDEX-scigen41.txt
INDEX-013.txt   is Genuine      0.61375975      data/samples/Scigen/INDEX-scigen25.txt
```

B.3.4 Max-Min text length

```
# the maximum, minimum size of a text
Max_length        30000
Min_length        10000
```

This set the max(min) length in characters (including white space) for a text to be eligible for classification. This parameter is used to avoid miss classification: when an article is too long, this causes the characteristics of the article to become too generic and a very long paper may be misclassified (without splitting misclassification rate: 0.13% or 42 misclassification/ 31577 samples). When the article is shorter than the Min length, it will be marked as cannot be classified.

The default value for max length is set at 30000 characters (about 10 pages); a longer text will be split into several parts which are tested individually. Default min length is set at 10000 characters.

Appendix C

SciDetect Web Service

A web service version of SciDetect is also provided and will be presented in this chapter.

C.1 Installation Requirements and Usage

SciDetect is also provided as a web service. It includes two distinct parts: a web application that needs to be deployed on a Tomcat server, and a client that can be used to call the process and given responses to the user.

C.1.1 Web Application

The Java web application implementing the web service requires Apache Tomcat 7 and Java SE 6 or higher.

The web application and all required runtime libraries are contained in the deployment package file `SciDetectServerXX.war`, which must be deployed on a Tomcat server.

The web application caches some of its data in a temporary directory (`/tmp/tomcat7_tmp`) and should be cleaned periodically.

C.1.2 Web Service Client

A client for the SciDetect web service is implemented in `SciDetectClient.jar` and can be used as a stand-alone Java program. The client component requires Java SE 6 or higher, no additional libraries are needed; However, the configuration file (configClient.txt) is required by the client.

C.1.3 Usage

The SciDetect web service client can be used in the same manner as the SciDetect local (with the same parameters), Please see Section B.2.

C.2 Configuration

A configuration file (configClient.txt and configServer.txt) is included with both the Server and the Client.

C.2.1 Client Configuration

The configuration file for the client (configClient.txt) should be found in the same directory with the SciDetectClient.jar and it contains:

Endpoint service

```
# Endpoint service location
Endpoint_Service          http://lexicometrie.imag.fr/SciDetectServer2.2/Checker?wsdl
```

This line is used to point the client to where the SciDetectServer is located, normally it is in the form of:

```
http://<host:port>/SciDetectServer2.2/Checker?wsdl
```

Threshold configuration & Path for log files

These configurations are the same as for Scidetect Local, please refer to Section B.3.

C.2.2 Server Configuration

The configuration file for the server should be found in the following directory on a Tomcat server along with the data directory:

`<path_to_tomcat>/webapps/SciDetectServer2.0/WEB-INF/`

It contains:

- Path to sample folder
- Max-Min text length

And can be configured the same in Section B.3

Appendix D

Extra information

D.1 Make use of detail logging

The detail log (parameter -d) stores all the distances from the text under test to all other samples in the sample set (i.e. all texts in all directories found at `/data/sample`). This can be used to get a more detailed look at the results.

For example: An article returned with a distant to the nearest neighbour that barely passes the threshold. Turning on the detail log for that article and checking the results may help in the decision.

```
INDEX-053.txt   data/samples/Mathgen/INDEX-mathgen55.txt        0.6821885795569994
INDEX-053.txt   data/samples/Mathgen/INDEX-mathgen63.txt        0.6608131367167517
INDEX-053.txt   data/samples/Scigen/INDEX-scigen36.txt  0.39296257670516693
INDEX-053.txt   data/samples/Mathgen/INDEX-mathgen9.txt 0.6679829987841077
INDEX-053.txt   data/samples/Scigen/INDEX-scigen0.txt   0.35342658461094817
INDEX-053.txt   data/samples/Mathgen/INDEX-mathgen47.txt        0.660816573503142
INDEX-053.txt   data/samples/Scigen/INDEX-scigen52.txt  0.3808927385660057
INDEX-053.txt   data/samples/Mathgen/INDEX-mathgen71.txt        0.6897595647595604
INDEX-053.txt   data/samples/Scigen/INDEX-scigen28.txt  0.38955875898790254
INDEX-053.txt   data/samples/Scigen/INDEX-scigen60.txt  0.39994884474379633
INDEX-053.txt   data/samples/Mathgen/INDEX-mathgen39.txt        0.6868800914402744
INDEX-053.txt   data/samples/Physgen/INDEX-physgen81.txt        0.5303053819516341
INDEX-053.txt   data/samples/Propgen/INDEX-17-html.txt  0.7981193467108959
INDEX-053.txt   data/samples/Physgen/INDEX-physgen65.txt        0.510647010647008
INDEX-053.txt   data/samples/Propgen/INDEX-53-html.txt  0.7880669668830156
INDEX-053.txt   data/samples/Physgen/INDEX-physgen5.txt 0.5160079114941755
INDEX-053.txt   data/samples/Physgen/INDEX-physgen73.txt        0.5115960731657623
INDEX-053.txt   data/samples/Physgen/INDEX-physgen49.txt        0.5055891144600811
INDEX-053.txt   data/samples/Propgen/INDEX-86-html.txt  0.7643301386956208
INDEX-053.txt   data/samples/Physgen/INDEX-physgen96.txt        0.5069873754844876
INDEX-053.txt   data/samples/Propgen/INDEX-45-html.txt  0.7918353315721742
INDEX-053.txt   data/samples/Scigen/INDEX-scigen21.txt  0.38484926003355824
INDEX-053.txt   data/samples/Mathgen/INDEX-mathgen78.txt        0.6692076400040969
INDEX-053.txt   data/samples/Propgen/INDEX-0-html.txt   0.7876861141791592
INDEX-053.txt   data/samples/Mathgen/INDEX-mathgen16.txt        0.682802115990133
INDEX-053.txt   data/samples/Physgen/INDEX-physgen10.txt        0.5261174636174665
```

As in the example, The file under-test has close distance to many of the SCIgen papers. This would make it highly likely to be automatically generated when compare to the case it is only closely related to one or two Scigen papers.

D.2 Tuning/Setting Thresholds

Thresholds for the current known generators have been empirically set according to tests presented in this section. These tests involve the computation of the intertextual distance presented in [1].

For each generator (Scigen, scigen-physics, Mathgen and propgen) a set of 400 texts is used (i.e: 1600 texts for the whole). For each text the distance to its nearest neighbour in the sample set is computed. The sample is composed of an extra 100 texts per generator (i.e: 400 additional texts). The nearest neighbour is always of the same nature than the tested text and columns 1-2-3-4 of Table 3.1 show statistical information about the observed distances.

A set of 8200 genuine papers is also used. For each genuine text the distance to its nearest fake in the sample set is computed. The sample still being composed of the same 400 texts (100 per generator). For each of the 8200 genuine papers, the nearest fake neighbour is in one of the generated sample group.

The first 2 rows of Table D.1 shows that, for a genuine paper, the minimal distance to the nearest fake is always greater than the maximal distance of the nearest neighbour of a fake.

Table D.1: Mean, min-max distances between papers and theirs nearest neighbour, along with standard deviation and median.

	Scigen	scigen-physics	Mathgen	Propgen	Genuine
Min distance to NN	0.30	0.31	0.19	0.11	0.52
Max distance to NN	0.40	0.39	0.28	0.22	0.99
Mean distance to NN	0.35	0.35	0.22	0.14	0.69
Standard deviation	0.014	0.012	0.014	0.015	0.117
Median	0.35	0.35	0.22	0.14	0.64

Scigen (`http://pdos.csail.mit.edu/scigen/` (dir `data/samples/SCIgen`)) The graph D.1 shows the observed distribution for texts having a Scigen text as nearest fake neighbour.

scigen-physics `https://bitbucket.org/birkenfeld/scigen-physics` (dir `data/samples/Physgen`) The graph D.2 shows the observed distribution for texts having a scigen-physics text as nearest fake neighbour.

Mathgen `http://thatsmathematics.com/mathgen/` (dir `data/samples/Mathgen`) The graph D.3 shows the observed distribution for texts having a mathgen text as nearest fake neighbour.

propgen `http://www.nadovich.com/chris/randprop/` (dir `data/samples/Propgen`) The graph D.4 shows the observed distribution for texts having a randprop text as nearest fake neighbour.

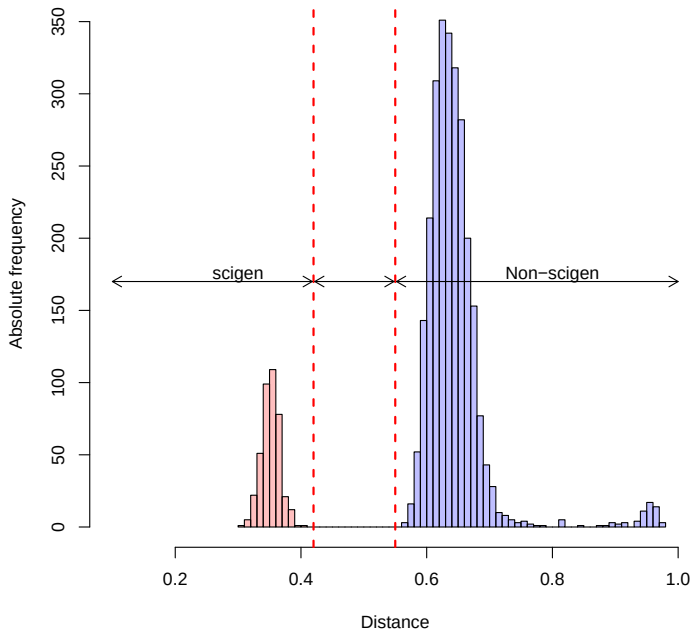


Figure D.1: Distribution of distances to the *S*cigen nearest neighbour. In blue for a set of *n*on-scigen paper. In red for a set of *s*cigen papers

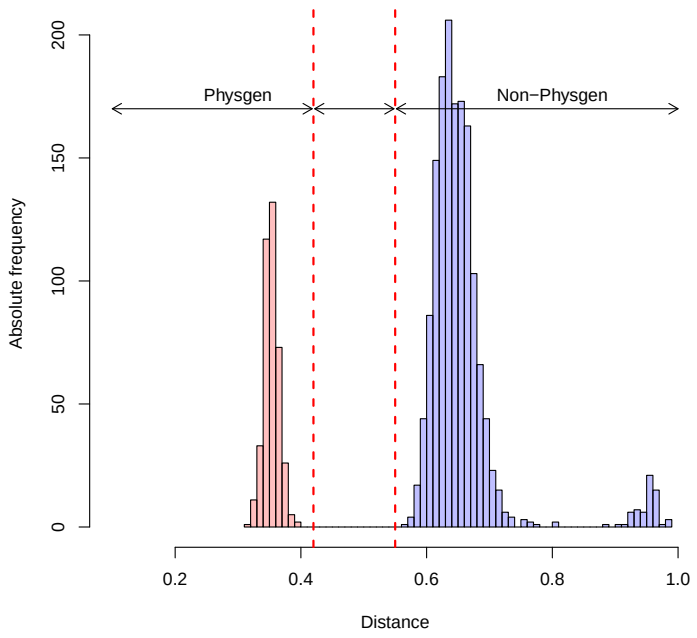


Figure D.2: Distribution of distances to the *s*cigen-physics nearest neighbour. In blue for a set of *n*on-scigen-physics paper. In red for a set of *s*cigen-physics papers

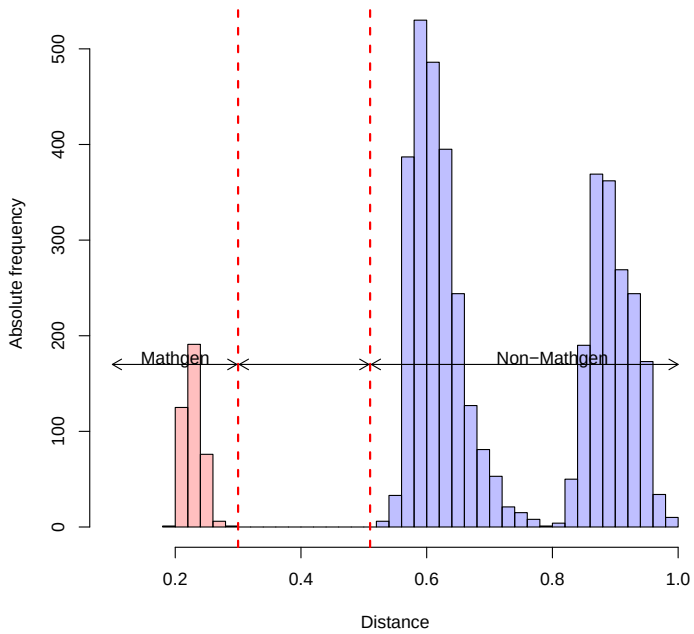


Figure D.3: Distribution of distances to the *mathgen* nearest neighbour. In blue for a set of *non-mathgen* paper. In red for a set of *mathgen* papers

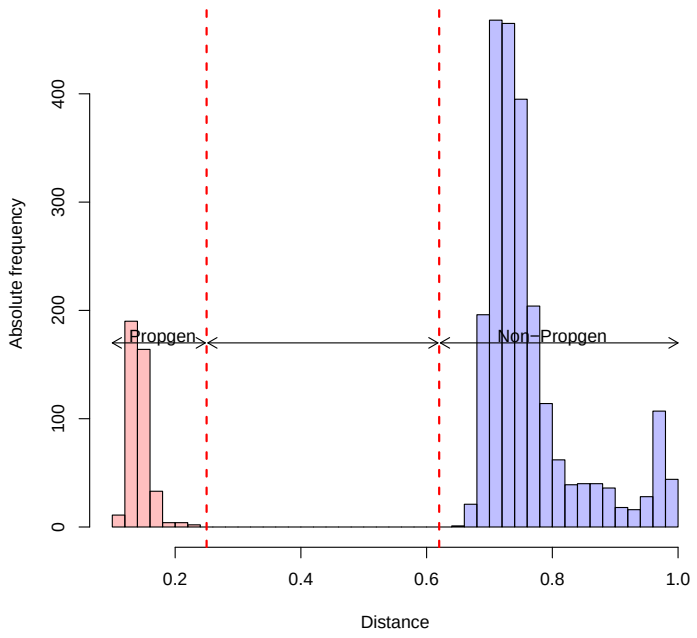


Figure D.4: Distribution of distances to the *randprop* nearest neighbour. In blue for a set of *non-randprop* paper. In red for a set of *r*andprop papers

Bibliography

[1] Cyril Labbé, Dominique Labbé. *Duplicate and fake publications in the scientific literature: how many SCIgen papers in computer science?* Scientometrics 94, no. 1 (2013): 379-396 (http://hal.archives-ouvertes.fr/hal-00641906v2/document).