Freie Universität Bozen
Libera Università di Bolzano
Free University of Bozen · Bolzano

Erasmus Mundus European Master in
Language & Communication Technologies (LCT)
Faculty of Computer Science
Free University of Bozen-Bolzano
EM European Masters Program in LCT
Université de Lorraine

Exploiting CCG Derivations within Distributional Semantic Models

Thesis Submission for a
Master of Science in
Computer Science

Paramita
Defense on
12^{th} October, 2012

Supervisor: Sergio Tessaris, Free University of Bozen-Bolzano
Co-Supervisor: Raffaella Bernardi, University of Trento
Co-Supervisor: Roberto Zamparelli, University of Trento

Acknowledgements

First and foremost, I wish to express my greatest gratitude to my supervisors who have helped and supported me throughout the completion of this thesis: Sergio Tessaris, for always being available to give his support and advice; Roberto Zamparelli, who helped a lot during the early stages of understanding the main topic of this thesis and finding the solutions for the arisen problems; and special thanks to Raffaella Bernardi, for her continuous support and guidance that make it possible for me to finish my thesis on time and for providing recommendations and ideas to enhance the quality of this thesis. I would also like to thank Marco Baroni and COMPOSES team who give me insight about distributional semantics and lead me to grow interest in broadening my knowledge in this area.

I would like to thank the Erasmus Mundus LCT consortium for giving me the opportunity to pursue my master degree and for the financial support that has made my study in Nancy and Bolzano possible.

I also wish to express my warm and sincere thanks to all of my comrade LCT students and classmates in Nancy and Bolzano for their company during this two years of study. Thanks to them, being a student was a great experience.

Finally, I am very thankful to my family and friends in Indonesia, Nancy, and Bolzano for their endless support and encouragement. It would not have been possible to write this thesis without all of the help and support of the kind people around me.

Abstract

For the last decade, Distributional Semantic Models (DSMs) have been an active area of research to address the problem of understanding the semantics of words in natural language. The central assumption which functions as the core of DSMs is that the linguistic context surrounding a given word provides important information about its meaning, and hence word co-occurrence statistics can provide a natural basis for semantic representations.

The main idea behind this work is to explore ways to incorporate syntactic information within the distributional semantic models. This approach is known as structured DSMs, as opposed to unstructured DSMs which use a set of unordered words as linguistic context without any syntactic information.

There have been some works focusing on structured DSMs which use syntactic categories such as part-of-speech (POS) tags (*i.e.* nouns, verbs, adjectives, etc.) and dependency relations between words (*i.e.* subject, direct object, modifier, etc.), to enrich the basis elements of context vector used to represent the meaning of a word in DSMs. Although it has been shown that dependency paths are a good approximation to semantic relations between words, the lack of information about syntactic category of each word in the context might become the shortcoming to fully exploit syntactic information within DSMs.

To this end, we exploit Combinatory Categorial Grammar (CCG) categories that provide a transparent relation between syntactic category and semantic type of the linguistic expression, to enrich the linguistic context used to represent the word meaning in DSMs. CCG categories, which are regarded as supertags, are supposed to be better at defining semantics than the traditional less informative POS tags since they carry more rich syntactic information.

We use the standard framework to build DSMs, which includes defining linguistic context, building the co-occurrence matrix, and applying various weighting schemes to the produced co-occurrence matrix. However, in order to build CCG-based DSM, constructing the co-occurrence matrix requires the use of CCG parsed corpus to extract the CCG categories of each word, hence full CCG parsing is carried out beforehand.

The constructed CCG-based DSM is then evaluated on one of widely known semantics tasks, which is word categorization, by comparing it with other types of DSM. From the experiments we could conclude that CCG-based DSM is generally better than POS-based DSM, and most certainly outperforms unstructured DSM that do not provide any syntactic information in the linguistic context. It is also shown that CCG-based DSM performs better than dependency-based DSM in the case of verb categorization task, demonstrating the importance of syntactic categories in defining the meaning of verbs.

We also investigate the impact of employing different context window on the performance of CCG-based DSM. Finally, we explore the effect of including the function words, which are grammatical words such as determiner, preposition, pronoun, etc. belonged to the group of closed-class words, in the co-occurrence matrix since existing works covering this topic usually use only content words including nouns, verbs, adjectives, and most adverbs.

Key words: Distributional Semantic Models, Combinatory Categorial Grammar, Word Categorization

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"You shall know a word by the company it keeps."
–John Rupert Firth, 1957

1.1 Distributional Semantics

Understanding the meaning of human language has been the ultimate goal in computational linguistics field, since the possibility of building computational models of linguistic behaviour relies heavily on the knowledge about meaning. A major issue for the study of meaning in a language is to devise precise identity criteria for the semantic content of the smallest unit in spoken language, the morpheme, which in many instances is itself a word.

We cannot proceed to profoundly investigate word meaning without specifying under which conditions two words can be said to have the similar meaning, or in other words to be semantically similar. Therefore, semantic similarity has a crucial role in any linguistic and psychological investigation on meaning. The degree of semantic similarity between words affect how they are processed or stored in the mental lexicon (Lenci, 2008).

It has been shown that for words in the same language, the more often two words can be substituted into the same contexts the more similar in meaning they are judged to be (Miller & Charles, 1991). This phenomenon that words that occur in similar contexts tend to have similar meanings has been widely known as *Distributional Hypothesis* (DH) (Harris, 1954), which can be stated in the following way:

Distributional Hypothesis
The degree of semantic similarity between two linguistic expressions A and B is a function of the similarity of the linguistic contexts in which A and B can appear.

This hypothesis is the core behind the application of vector-based models for semantic representation, which are variously known as word space (Sahlgren, 2006), semantic spaces (Mitchell & Lapata, 2010), vector spaces (Turney & Pantel, 2010), corpus-based semantic models, or, the term used by Baroni and Lenci (2010) that we will adopt, distributional semantic models (DSMs).

To have better illustration about distributional hypothesis, consider a foreign word such as *wampimuk*, occurring in these two sentences: (1) *He filled the wampimuk, passed it around and we all drunk some*, and (2) *We found a little, hairy wampimuk sleeping behind the tree.* We could infer that the meaning of *wampimuk* is either 'cup' or 'animal', heavily depends on its context which is either sentence (1) or (2) respectively.

In distributional semantics approach, capturing the semantic properties of words is as simple as building a multi-dimensional space by vectors that are constructed from large bodies of text, by observing the distributional patterns of co-occurrence with their neighboring words. Co-occurrence information is typically collected in a frequency matrix, where each row corresponds to a unique word, commonly referred to as "target word" and each column represents a given linguistic context (Padó & Lapata, 2007).

The advantage of taking such a geometric approach is that the semantic similarity between any two words can be easily quantified by measuring their Euclidean distance in the vector space, or the cosine of the angle between them, as demonstrated in Figure 1.1.

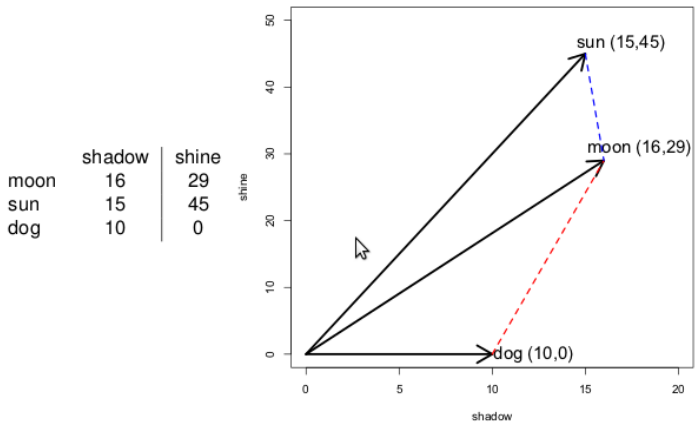


Figure 1.1: The geometry of word meaning

According to Turney (2006) there are at least two kinds of semantic similarity, *attributional similarity* and *relational similarity*. The word–context matrices previously mentioned are well suited for measuring *attributional similarity*, which depends on the degree of correspondence between the properties of two words. The more correspondence there is, the greater their attributional similarity. Examples are synonyms (*rich* and *wealthy*), meronyms (*car* and *wheel*), antonyms (*hot* and *cold*), words that are functionally related or frequently associated (*pencil* and *paper*), and words that share a hypernym (*dog* and *cat*, because they share the hypernym *animal*). We might not usually think that antonyms are similar, but antonyms have a high degree of attributional similarity, for example *hot* and *cold* might share the same attributes related to temperature.

Whereas *relational similarity* between two *pairs* of words, namely *A:B* and *C:D*, is defined as the degree of correspondence between the relations between *A* and *B* and the relations between *C* and *D* (Turney, 2006). For example, pair of words *dog:animal* and *car:vehicle* are linked by similar semantic relations, which is hypernymy. When two words have a high degree of attributional similarity, we call them *synonyms*. However when two pairs of words have a high degree of relational similarity, we say that their relations are *analogous*.

To accommodate the relational similarity problem, pair–pattern matrix was introduced. Row vectors correspond to pairs of words, such as *mason:stone* and *carpenter:wood*, and column vectors correspond to the patterns in which the pairs co-occur, such as *"X cuts Y"* and *"X works with Y"*. The extended distributional hypothesis proposed by Lin and Pantel (2001) then holds, that patterns that co-occur with similar pairs tend to have similar meaning. Pattern similarity can be used to infer that one sentence is a paraphrase of another.

Although semantic similarity, either attributional or relational, play a large part in DSMs, similarity is not the only aspect of meaning that is addressed by distributional approaches.

The wide variety of semantic issues that DSMs are able to tackle, which will be discussed later, confirms the importance of looking at distributional data to explore meaning.

1.2 Success Stories

The idea of representing word meaning by distributional semantics have become increasingly popular in cognitive science, motivated by the success of vector space models (VSMs) for information retrieval. The basic idea is to represent each document in a collection as a point in a space, namely a vector in a vector space. Points that are close together in this space are perceived as semantically similar and points that are far apart are semantically distant.

Vectors have been a common models in artificial intelligence and cognitive science, even before VSM was introduced for information retrieval system. In machine learning, a typical problem is to learn to classify or cluster a set of items represented as *feature vectors*. A feature vector is an n-dimensional vector of numerical features that represent some object. When representing images, the feature values might correspond to the pixels of an image, and perhaps term occurrence frequencies for representing text, as what is done in DSMs. Most of the VSM-based systems achieve state-of-the-art performance in text classification or categorization.

There are a number of well-known DSMs in the literature. Latent Semantic Analysis (LSA) (Landauer & Dumais, 1997) is an example of document-based vector space model that is commonly used in cognitive science. LSA assumes that words that are close in meaning will occur in similar pieces of text. LSA constructs a word–document co-occurrence matrix from a large document collection, where each rows represent target words and columns represent each paragraph they occur in. In contrast, the Hyperspace Analogue to Language model (HAL) (Lund & Burgess, 1996) creates a word-based vector space model. Each word is represented by a vector where each element of the vector corresponds to a weighted co-occurrence value of that word with some other word within a certain window size.

DSMs have been successful at simulating a wide range of psycholinguistic phenomena, especially tasks which require a measure of semantic similarity, including semantic priming (Lund & Burgess, 1996; Landauer & Dumais, 1997; Griffiths, Tenenbaum, & Steyvers, 2007), word association (Denhière & Lemaire, 2004; Griffiths et al., 2007), word sense disambiguation (Schütze, 1998), word categorization or classification (Laham, 2000; Turney & Littman, 2003), or synonym tests such as those included in the Test of English as Foreign Language (TOEFL) (Landauer & Dumais, 1997; Griffiths et al., 2007). On the last task mentioned, Rapp (2003) used a vector-based representation of word meaning to achieve a score of 92.5% on multiple-choice synonym questions, whereas the average human score was 64.5%.

DSMs have also found wide applications in computational lexicography, especially for automatic thesaurus construction (Crouch, 1988; Grefenstette, 1994; Pantel & Lin, 2002; Curran & Moens, 2002; Rapp, 2004). Consider WordNet, a lexical database for the English language (Miller, Beckwith, Fellbaum, Gross, & Miller, 1990). It groups English words into sets of synonyms called *synsets*, provides short, general definitions, and records the various semantic relations between these synonym sets. WordNet can be seen as a combination between dictionary and thesaurus and is a popular tool for research in natural language processing. However, creating and maintaining such lexical resources require a great effort, so it is natural to wonder whether the process can be automated to some degree.

Automatic thesaurus generation can be seen either as word clustering task, when the thesaurus is generated from scratch, or as classification task, to automatically extend already existing lexical resources. DSMs are widely used for the semi-automatic bootstrapping or extension of terminological repositories, computational lexicons such as WordNet, and ontologies (Buitelaar & Magnini, 2005; Lenci, 2010).

Another attractive point of DSMs is that they extract knowledge automatically from a given corpus, thus require much less labour than other approaches to semantics, such as hand-coded knowledge bases and ontologies. Although a considerably huge corpus is needed to build a representative knowledge, gathering a corpus for a new language is generally much easier than building a lexicon.

Many researchers believe that DSMs are plausible to model some aspects of human cognition. Such plausibility is not essential, but it may be seen as a sign that DSMs are a promising area for further research.

1.3 Research Questions

If we focus on investigating the attributional similarity of words, which will be the concentration of this thesis, the main idea of distributional semantic models is to build co-occurrence matrix where each row corresponds to a *target word* and each column represents a *linguistic context*. The important issue is what to take as the linguistic context that may draw the origin and format of semantic representations in the human mind.

The simplest way would be to represent context as a set of unordered words, without even taking parts-of-speech into account. The construction of semantic models hence is straightforward and language-independent, since all that is needed is a segmented corpus of written or spoken text. Baroni and Lenci (2010) refers to such models as *unstructured DSMs*, because they do not use the linguistic structure of texts to compute co-occurrences.

Structured DSMs on the other side are built on the assumption that contextual information alone is not enough to define a word's meaning and it is clearly a simplification, hence explaining the efforts to enrich DSMs with syntactic information (Padó & Lapata, 2007).

There have been some works which focus on syntactically enhanced distributional vectors, embedding part-of-speech (POS) information in the co-occurrence matrix (Kanejiya, Kumar, & Prasad, 2003; Widdows, 2003). Furthermore, numerous studies have been carried out to investigate the impact of exploiting syntactic relation based on dependency grammar in DSMs (Grefenstette, 1994; Lin, 1998; Curran & Moens, 2002; Padó & Lapata, 2007; Rothenhäusler & Schütze, 2009; Baroni & Lenci, 2010).

The choice of linguistic context is one of the interesting research questions in DSMs. Despite the debate on the costs and benefits of having structured contexts, the unlimited possibilities to enrich the distributional statistics with syntactic information leave the door open for researchers to explore ways of combining them in distributional semantics approach.

Related with defining the linguistic context, the second issue in constructing DSMs is deciding the size of context window. The size of context window, commonly addressed with the term *window size*, determines the maximum distance of context words from target words to be called as co-occur together. Bullinaria and Levy (2007) investigates the outcome of varying the window size with respect to the performance of DSMs. The change in performance as one varies the window size can be understood as a consequence of the trade-off of the increased context information for larger windows, against the increased likelihood of irrelevant and misleading context information.

The other issue is that the construction of linguistic context usually includes the removal of function (or grammatical) words, under the assumption that only the content words would determine the contextual meaning of a target word. *Function words* belong to closed class of words (only about 300 in English), comprises prepositions, pronouns, determiners, conjunctions, particles, modal verbs, and auxiliary verbs. Whereas the *content words* are open class words, which are usually nouns, verbs, adjectives, and adverbs.

Even though the idea of limiting the contextual meaning with only content words seems reasonable since these are the ones which give meaning to a text, the question is what will be the consequence of including function words in distributional semantic framework.

1.4 Contributions of the Thesis

The aforementioned research questions will be the starting point of the thesis and henceforth define the contributions of the thesis to the studies of distributional semantic models.

We are mainly interested in structured DSMs, since previous works that have been carried out suggest that structured models have slightly better performance than unstructured DSMs (Padó & Lapata, 2007; Rothenhäusler & Schütze, 2009). The argument is based on the fact that structured DSMs capture more linguistic structure than unstructured models, thus they should at least in theory provide more informative representations of word meaning.

As mentioned before, there have been some works which enrich the linguistic context with part-of-speech (POS) tag and dependency-based syntactic information. Although it has been shown that dependency paths are a good approximation to semantic relations between words, the lack of information about syntactic category of each word in the context might become the shortcoming to fully exploit syntactic information within DSMs.

The novel idea of this work is to employ *supertag* (Bangalore & Joshi, 1999) as opposed to the traditional POS tag. The idea behind supertagging is to extend the notion of 'tag' from a part-of-speech to a tag that represents rich syntactic information as well.

Several phrase structure-based grammar formalisms are exploiting the idea of doing supertagging before the actual parsing, one of them is Combinatory Categorial Grammar (CCG). This selection of choice is generally incited by the transparent relation between syntactic category and semantic type in CCG, which makes CCG category an appealing idea to provide the context with not only syntactic information but also its intrinsic type within semantics.

The contribution of this thesis is two-fold. First, we will introduce a general framework to build CCG-based DSM, based on the standard framework to build DSMs. Second, we will present an empirical studies which will be carried out on well-established semantics tasks to address the previously stated research questions as follows:

- comparison of our enhanced DSM with CCG categories against other type of DSMs including unstructured model, POS-based model, and state-of-the-art dependency-based model
- investigation regarding the impact of varying window size on the performance of CCG-based DSM
- investigation regarding the effect of the inclusion of function words in the linguistic context

1.5 Structure of the Thesis

The rest of the thesis is organized as follows:

Chapter 2 gives a general overview of various syntactic structures, and introduction to Combinatory Categorial Grammar (CCG) as one of the grammatical formalism which based on phrase structure grammar. The discussion about CCG includes the explanation about the CCG parser used in this thesis, C&C Parser, also the problems found within the parser and the proposed solutions. The general framework to build CCG-based DSM is also discussed in this chapter.

Chapter 3 provides an overview about the corpus used in the experiment, including the process to produce CCG parsed corpus, which will be used to build the distributional semantic models. This chapter also gives brief statistics regarding the CCG parsed corpus.

Chapter 4 discusses the evaluation tasks which is used to measure the performance of the CCG-based DSM, particularly compared with other models which perform the experiments

on the same evaluation tasks. The main task is word categorization, which is perceived as clustering task. The tool used for clustering, CLUTO, is then introduced. This chapter also discusses different methods to measure clustering quality.

Chapter 5 covers discussion about the practical steps required to build the CCG-based DSM, followed by various experiments that are carried out. The result of the experiments is then presented along with the analysis.

Finally, Chapter 6 sums up the conclusion drawn on the studies throughout the previous chapters. Some future research possibilities are also mentioned in this chapter.

Chapter 2

Background

2.1 Syntactic Structures

Syntax concerns the way that words are arranged into larger units. The largest unit that usually considered in syntactic analysis is the sentence. Analyzing a sentence, commonly known as *parsing*, is the process to build syntactic structures of a sentence. There are two major types of syntactic structure in linguistics studies, *phrase structure* and *dependency structure*.

Phrase structure model is constituency-based structure, which views sentence structure in terms of the constituency relation. The term phrase structure grammar, also known as context-free grammar (CFG), was introduced by Chomsky (1957) as the term for grammars that use phrase structure rules to break sentences down into their constituent parts. Phrase structure rules are usually in the form of $A \rightarrow B\ C$, meaning that the constituent A is separated into the two sub-constituents B and C. Hence, the combinations of constituents are often represented as tree structures, as shown in Figure 2.1.

On the contrary, dependency grammar (Tesnière, 1959) determines sentence structure based on dependency relation between individual words. The dependencies between words can be represented in various ways, as illustrated in Figure 2.2. Basically, the structure contains *head* and its *dependent*, and the *relation* between them which constitutes the dependency.

In general, a phrase structure representation may be found more suitable for languages with rather fixed word order patterns and clear constituency structures. While dependency representations may be found more adequate for languages which allow greater freedom of word order such as Czech and Turkish.

Linguists are quite sure that human language cannot be modeled with systems less than

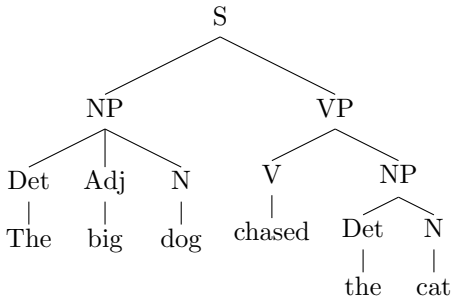


Figure 2.1: Phrase structure tree for the sentence *The big dog chased the cat.*

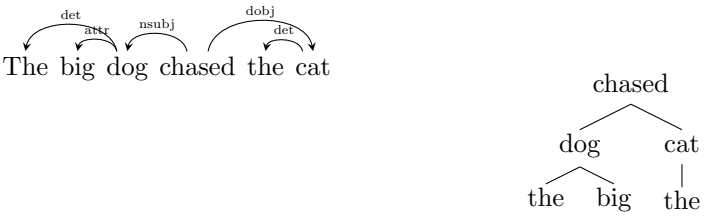


Figure 2.2: Dependency structure for the sentence *The big dog chased the cat.*

or equal in expressive power to context-free grammar. Therefore, there have been quite a number of grammar formalisms based on phrase structure grammar, which are more linguistically expressive. Some of them are Categorial Grammar (Bar-Hillel, 1953), Head-Driven Phrase Structure Grammar (HPSG) (Proudian & Pollard, 1985), Tree-Adjoining Grammar (TAG) (Joshi, 1985), and Lexical Functional Grammar (LFG) (Dalrymple, 2001). They usually include syntactic features of a language in the parsing, allowing them to capture complex linguistic phenomena such as ambiguity, coordination, and long range dependency. Thus, they are known as wide-coverage parsers.

Among them, TAG and Categorial Grammar are the least expressive generalization of context-free grammar known. They employ *supertag* (Bangalore & Joshi, 1999), extend the notion of 'tag' from a part-of-speech (POS) to a tag that represents rich syntactic information. Supertagging is done prior to parsing, leaving the parser with less work to do since the lexical categories are already provided with syntactic information. This characteristic makes them more powerful than CFG while remaining efficiently parsable.

2.2 Combinatory Categorial Grammar (CCG)

Categorial Grammar is one of the lexicalized grammatical formalism, in which grammatical constituents are distinguished by their syntactic types, or *categories*. These categories could be seen as either a function from arguments which transform one type into another, or as an argument. One thing that makes this particular grammatical formalism interesting, thus resulting in its development in early 1980's, is the transparent relation between syntactic category and semantic type of the linguistic expression.

Combinatory Categorial Grammar (CCG) (Steedman & Baldridge, 2011) is an extension of Categorial Grammar. CCG was designed to deal with long-range dependencies, which are relatively common in text such as newspaper text. As a lexicalized grammar, the application of syntactic rules is entirely based on the syntactic type, or *category*, of their inputs. No rule is structure- or derivation- dependent. This lexicalized nature of the grammar has implications for the engineering of a wide-coverage parser.

Figure 2.3 shows the CCG derivation for the sentence *The big dog chased the cat.* As mentioned before, categories identify constituents as either *primitive categories* or *functions*. Example of *primitive categories* are N (noun), NP (noun phrase), PP (prepositional phrase), S (sentence), and so on. Whereas *functions* takes arguments (may be either functions or primitive categories) to result in a category identifying the type. Take a transitive verb *chased* as an example, which falls into category $(S\backslash NP)/NP$, indicating that it needs noun phrase to its right and also its left.

In the original Categorial Grammar, there is only one composition relation: *functional application* that is given into two variants. The first rule is *forward application* ($>$) and the second rule is *backward application* ($<$).

$$X/Y \quad Y \Rightarrow X \quad (>) \tag{2.1}$$

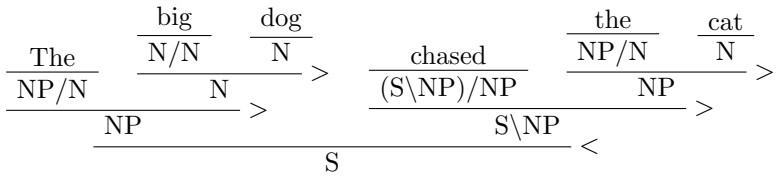


Figure 2.3: CCG derivation for the sentence *The big dog chased the cat.*

$$Y \quad X\backslash Y \Rightarrow X \quad (<) \tag{2.2}$$

In order to increase the expressivity of the grammar, CCG introduces further rules for combining categories which are called *combinatory* rules, giving it the name of Combinatory Categorial Grammar.

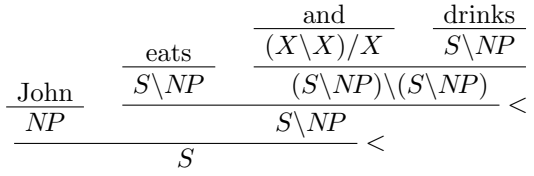
2.2.1 Combinatory Rules

Coordination

Coordination rule conjoins categories of the same type, by assuming the following category schema for conjunctions: $(X\backslash X)/X$, where X could be any category.

$$(X\backslash X)/X \quad X \Rightarrow (X\backslash X) \quad (coord.) \tag{2.3}$$

The coordination rule is basically forward application rule applied to category schema for conjunctions and any category X. It gives the following derivation for sentence *John eats and drinks*:



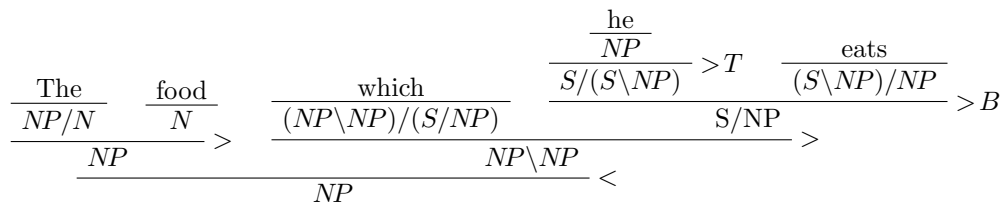
Composition

There are two functional composition rules, *forward composition* ($>B$) and *backward composition* ($<B$).

$$X/Y \quad Y/Z \Rightarrow X/Z \quad (>B) \tag{2.4}$$

$$Y\backslash Z \quad X\backslash Y \Rightarrow X\backslash Z \quad (<B) \tag{2.5}$$

Forward composition is often used together with *type-raising* ($>T$) that we will present in more details later. It allows the analysis of sentences or phrases such as *The food which he eats*:



Further composition rules in CCG include modified rule of forward composition which is called *forward crossed composition* ($>B_x$). Similarly, modified rule of backward composition called *backward crossed composition* ($<B_x$) is also defined.

$$X/Y \quad Y\backslash Z \Rightarrow X\backslash Z \quad (>B_x) \tag{2.6}$$

$$Y/Z \quad X\backslash Y \Rightarrow X/Z \quad (<B_x) \tag{2.7}$$

Backward crossed composition, which is applied more frequent than backward composition, can often be found in negative declarations that require conjoining modal verbs and 'not'. For example, in sentences such as *John does not eat*:

$$\dfrac{\dfrac{\text{John}}{NP} \quad \dfrac{\dfrac{\dfrac{\text{does}}{(S\backslash NP)/(S\backslash NP)} \quad \dfrac{\text{not}}{(S\backslash NP)\backslash(S\backslash NP)}}{(S\backslash NP)/(S\backslash NP)}<B_x \quad \dfrac{\text{eat}}{S\backslash NP}}{S\backslash NP}>}{S}<$$

Functional composition rules can be *generalized* to allow additional arguments to the right of the Z category in (2.4), (2.5), (2.6) and (2.7), up to a fixed degree N_B. The generalized composition rules are then defined as follows:

$$X/Y \quad Y/Z\$ \Rightarrow X/Z\$ \quad (>B^n) \tag{2.8}$$

$$X/Y \quad Y\backslash Z\$ \Rightarrow X\backslash Z\$ \quad (>B^n_x) \tag{2.9}$$

$$Y\backslash Z\$ \quad X\backslash Y \Rightarrow X\backslash Z\$ \quad (<B^n) \tag{2.10}$$

$$Y/Z\$ \quad X\backslash Y \Rightarrow X/Z\$ \quad (<B^n_x) \tag{2.11}$$

where \$ denotes the additional arguments. These generalized rules accommodate analysis of sentences such as *I offered and may give a flower to a policeman*:

$$\dfrac{\dfrac{\text{may}}{(S\backslash NP)/(S\backslash NP)} \quad \dfrac{\text{give}}{(S\backslash NP)/PP/NP}}{(S\backslash NP)/PP/NP}>B^2$$

Type-Raising

Combinatory grammars also include type-raising rules, which turn arguments into functions over functions-over-such-arguments. This rule is often applied together with forward composition, since it allows arguments to compose with the verbs that seek them.

$$X \Rightarrow T/(T\backslash X) \quad (>T) \tag{2.12}$$

where T is a metavariable over categories. If T is instantiated as S, then it allows such derivation that can be seen in 2.2.1.

2.2.2 C&C Parser

The C&C CCG parser is a statistical parser developed by James Curran and Stephen Clark (Bos, Clark, Steedman, Curran, & Hockenmaier, 2004). As a statistical parser, it needs training data to build the statistical models, which is provided by *CCGbank*.

Prior to parsing, it employs the use of *supertagger* to statistically assign a small number of lexical categories to each word in the sentence. Since there is so much syntactic information in lexical categories, the parser is required to do less work, which leads to become

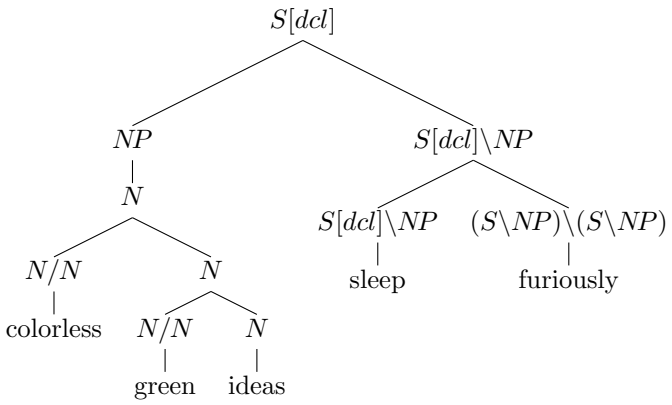


Figure 2.4: CCG derivation as binary tree for the sentence *Colorless green ideas sleep furiously.*

an extremely efficient parser. The parser is mentioned to be able to parse 20 sentences per second on standard hardware.

The C&C tools which can be found online[1] consist of the C&C CCG parser – including the computational semantics tool, Boxer – and the C&C taggers. The tools are released under an academic (non-commercial) licence.

CCGbank

CCGbank (Hockenmaier, 2003) is the CCG version of Penn treebank, which is created by converting the phrase-structure trees in the Penn Tree-bank into CCG normal-form derivations, as demonstrated in Figure 2.4. However, in order for the conversion to achieve a correct CCG analysis form some constructions, such as coordination, some pre-processing was required.

The basic algorithm to convert Penn treebank into CCGbank consists of three steps. First, the constituent type of each node (head (h), complement (c), or adjunct (a)) is determined. Then the flat trees are transformed to binary trees, which involves inserting dummy nodes into the tree. Finally, categories are assigned to the nodes in the binary tree, corresponding to a reverse CCG derivation.

The atomic categories considered in CCGbank are S, NP, N, and PP. It also employs features to distinguish between different kinds of sentences and verbs, as follows:

- Sentences:
  - $S[dcl]$: declarative sentences
  - $S[wq]$: wh-questions
  - $S[q]$: yes-no questions
  - $S[qemb]$: embedded questions
  - $S[emb]$: embedded declaratives
  - $S[frg]$: sentence fragments
  - $S[for]$: small clauses headed by *for*
  - $S[intj]$: interjections
  - $S[inv]$: elliptical inversion

[1] `http://svn.ask.it.usyd.edu.au/trac/candc`

- Verbs:
  - $S[b]\backslash NP$: bare infinitives, subjunctives and imperatives
  - $S[to]\backslash NP$: to-infinitives
  - $S[pass]\backslash NP$: past participles in passive mode
  - $S[pt]\backslash NP$: past participles used in active mode
  - $S[ng]\backslash NP$: present participles

In the CCG derivation, either the features have to be match, or one of the two is underspecification and the feature is then percolated up. As an example, which can be seen in Figure 2.4, the combination between $S[dcl]\backslash NP$ and $(S\backslash NP)\backslash(S\backslash NP)$ resulting in $S[dcl]\backslash NP$.

Besides used as training data for the statistical models, CCGbank also provides the lexical category set to be used by the supertagger, plus some unary *type-changing* rules and punctuation rules used by the parser.

In Figure 2.4 we can see a common type-changing rule in CCGbank, which changes a noun category N into a noun phrase NP. A complete list of the unary type-changing rules used by the parser is provided in (Bos et al., 2004).

Supertagger

The process of parsing with lexicalized grammatical formalism such as CCG requires two steps: assigning lexical categories to each word in the sentence, then combining the structures together. The simplest way to do the first step is by assigning to each word all the lexical categories which the word is assigned with in the training data, together with some strategy to deal with unknown words (perhaps assigning the complete set of the lexical categories).

Since the vector of lexical categories assigned to a word can be very big, some strategy is needed to make the parser run efficiently. In this case, supertagger approach is employed. Supertagger is described as using log-linear models to define a distribution over the lexical category set for each local 5-word context containing the target word (Clark & Curran, 2004).

The vector features used in the models consist of words and POS tags in the 5-word window, plus two previously assigned lexical categories to the left. The conditional probability of a sequence of lexical categories, given a sentence, is defined as the product of the probability of each category. The most probable lexical category sequence can be found using Viterbi algorithm for HMM taggers.

The number of categories assigned to a word is restricted by *tag dictionary*:

- for words seen at least k times in the training data, the tagger can only assign categories which have been seen assigned to the word.
- for words seen less than k times in the training data, the tagger can only assign categories which have been seen assigned to the word with the same POS tag.

A value $k = 2$ is used in the experiment. This strategy of restricting the selection of categories is proven to be beneficial to the parsing process, in terms of efficiency and accuracy.

The accuracy of the supertagger on section 00 of CCGbank is 92.6%, with a sentence accuracy of 36.8%. Sentence accuracy is the percentage of sentences whose words are all tagged correctly (Bos et al., 2004).

Combining Supertagger and Parser

At first, the strategy to combine the supertagger and parser was to use a non-restrictive setting of the supertagger, but still allow a reasonable compromise between speed and accuracy (Clark & Curran, 2003). However, the parsing process was so slow for some sentences, because the number of lexical categories was extremely large.

Hence the opposite approach was taken, lexical categories assigned by the supertagger with a very restrictive setting. The parser then uses the grammar to decide whether the categories provided by supertagger are acceptable, and if not the parser will request more categories. Using this adaptive supertagging approach, the parsing speed improves significantly without any corresponding loss in accuracy.

The algorithm that is used by the parser to build the packed charts is the CKY chart parsing algorithm, which applies naturally to CCG since the grammar is binary.

2.2.3 Problems with *NP* Interpretation

One of the evident problem occurs with the current version of C&C parser is the way the parser treated *relative pronouns*, i.e.: who, whom, whose, which, and that. As can be seen in the CCG derivation of the phrase *The food which he eats* (see 2.2.1), relative pronoun *which* is treated as an *NP* modifier, modifying *The food* (*NP*).

Whereas according to linguistic theory, relative pronouns should be treated as an N modifier, bringing the clause *which he eats* to be attached to the *food* (N) instead:

$$\dfrac{\dfrac{\text{The}}{NP/N}\quad\dfrac{\dfrac{\text{food}}{N}\quad\dfrac{\dfrac{\text{which}}{(N\backslash N)/(S/NP)}\quad\dfrac{\dfrac{\dfrac{\text{he}}{NP}}{S/(S\backslash NP)}>T\quad\dfrac{\text{eats}}{(S\backslash NP)/NP}}{S/NP}>B}{N\backslash N}<}{N}>}{NP}>$$

This kind of problem also occurs with other adnominals such as *prepositional phrases*, producing non-restrictive interpretations (Honnibal, Curran, & Bos, 2010). The CCG derivation of the phrase *All staff on casual contracts* below will produce wrong interpretation that there were no permanent staff.

$$\dfrac{\dfrac{\dfrac{\text{All}}{NP/N}\quad\dfrac{\text{staff}}{N}}{NP}>\quad\dfrac{\dfrac{\text{on}}{(NP\backslash NP)/NP}\quad\dfrac{\dfrac{\dfrac{\text{casual}}{N/N}\quad\dfrac{\text{contracts}}{N}}{N}>}{NP}TC}{NP\backslash NP}>}{NP}<$$

2.2.4 Optimized C&C Parser for Improved *NP* Interpretation

These problems have been solved by Matthew Honnibal, James R. Curran and Johan Bos, together with other related issues, by *rebanking* the CCGbank (Honnibal et al., 2010).

Treebanking is a tedious process, and usually despite any shortcomings in linguistic analysis occur in the treebanks, they tend to remain unchanged. The treebank however has been proved could be rebanked, which includes integrating new resources and implementing improved analysis for specific constructions.

The parser is then retrained with the rebanked CCGbank. The rebanked CCGbank includes improvement for restrictivity distinction in the adnominals to fix the problem previously explained in section 2.2.3, and also other improvements stated below. Thus, this optimized version of the parser has been chosen as the tools for CCG parsing in this project.

Adnominals Restrictivity Distinction

To solve the problem, punctuation is used to make the attachment decision. All $NP\backslash NP$ modifier that are not preceded by punctuation were moved to the lowest N node possible and relabelled $N\backslash N$.

Some adnominals in CCGbank are created by the $S\backslash NP \rightarrow NP\backslash NP$ unary type-changing rule, which transforms reduced relative clauses. Rule $S\backslash NP \rightarrow N\backslash N$ is then introduced in its place, and a binary rule cued by punctuation is added to handle the relatively rare non-restrictive reduced relative clauses.

The rebanked corpus contains 34,134 $N\backslash N$ restrictive modifiers, and 9,784 non-restrictive modifiers. Most (61%) of the non-restrictive modifiers were relative clauses.

Compound Noun Brackets

In the process of acquiring CCGbank from the Penn Treebank, bracketing ambiguities in compound noun phrases posed a problem, because the structure of such phrases is left underspecified in the Penn Treebank.

In CCGbank the derivations must be binary branching, so it is not possible to leave such structures underspecified. A simple heuristic then was employed: assume all such structures branch to the right. This would lead to incorrect analysis in the case of *(crude oil) prices* vs *crude (oil prices)*.

As a solution, all of the ambiguous noun phrases in the Penn Treebank are manually annotated, and this information is then used to correct 20,409 dependencies in CCGbank.

Punctuation Corrections

Punctuation is not always treated consistently in the Penn Treebank, and quotation marks were particularly problematic so they decided to remove them from CCGbank (Hockenmaier, 2003). As the correction, the quotation marks is restored and commas are shifted so that they always attach to the constituent to their left.

Verb Predicate-argument Corrections

In linguistics theory there is the distinction between adverbial adjunct and adverbial complement. This distinction is represented in CCG, because the category of a verb must specify its argument structure.

He	*joined*	*as a striker*	*vs*	*He*	*joined*	*as a striker*
NP	$(S\backslash NP)/PP$	*PP*		*NP*	$S\backslash NP$	$(S\backslash NP)\backslash(S\backslash NP)$

In the first one, *as a striker* is considered to be a complement, whereas in the second one as the adjunct. CCGbank contains noisy complement and adjunct distinctions, because in the Penn treebank this distinction is imperfectly represented. Using Propbank (Palmer, Gildea, & Kingsbury, 2005), 1,543 complements were converted into adjuncts, and 13,256 adjuncts into complements.

Verb-particle Constructions

In the CCGbank all intransitive prepositions are annotated as adjuncts, because the annotation of verb-particle constructions is not available in the Penn treebank. Using the Propbank, annotation to such constructions is added, by introducing new category PR for particles and changing their status from adjuncts into complements.

He	$woke$	up	$into$	He	$woke$	up
NP	$S\backslash NP$	$(S\backslash NP)\backslash(S\backslash NP)$		NP	$(S\backslash NP)/PR$	PR

Noun Predicate-argument Structure

Similar to verb predicate-argument structure explained in section 2.2.4, there is distinction between complements and adjuncts as the arguments in the nouns structure, which are realised by prepositional phrases, genitive determiners, compound nouns, and relative clauses.

$$\dfrac{\dfrac{\text{destruction}}{N/PP_y} \quad \dfrac{\dfrac{\text{of}}{PP_y/NP_y} \quad \dfrac{\text{Carthage}}{NP}}{PP_{Carthage}}>}{N_{destruction}}>$$

$$\dfrac{\dfrac{\text{war}}{N} \quad \dfrac{\dfrac{\text{in}}{(N_y\backslash N_y)/NP_z} \quad \dfrac{\text{149 B.C.}}{NP}}{(N_y\backslash N_y)_{in}}>}{N_{war}}<$$

For arguments introduced by prepositions the distinction can be applied quite straightforwardly as presented in above example[2]. In the first case, *of Carthage* functions as complement and therefore the head is *Carthage*. While in the second one, adjunct prepositional phrase *in 149 B.C.* remains headed by the preposition, since it is the preposition that determines whether they function as temporal, spatial, etc. argument.

$$\dfrac{\dfrac{\text{Carthage}}{NP} \quad \dfrac{\dfrac{\text{'s}}{(NP_y/(N_y/PP_z)_y)\backslash NP_z}}{(NP_y/(N_y/PP_{Carthage})_y)_{'s}}< \quad \dfrac{\text{destruction}}{N/PP_y}}{NP_{destruction}}>$$

$$\dfrac{\dfrac{\text{Rome}}{NP} \quad \dfrac{\dfrac{\text{'s}}{(NP_y/N_y)\backslash NP_z}}{(NP_y/N_y)_{'s}}< \quad \dfrac{\text{gift}}{N}}{NP_{gift}}>$$

For arguments introduced by possessives, the distinction is achieved by having the noun subcategories for the argument, which is typed PP, as can be seen in above example[2].

The analysis of distinction between these two arguments requires semantic role labels for each argument of the nominal predicates in the Penn Treebank, and this is provided by NomBank (Meyers et al., 2004). NomBank follows the same format as Propbank, so the procedure is exactly the same. 34,345 prepositional phrases are converted into complements, leaving 18,919 as adjuncts. There are 20,250 possessives in the corpus, of which 75% were converted to complements.

[2]Words as categories' head in the examples are only there for explanatory reason.

2.3 CCG for Distributional Semantic Models

2.3.1 Defining the Linguistic Context

The process of defining the linguistic context involves choosing between unstructured or structured models as previously discussed in Section 1.3, and deciding the window size (i.e. the number of words surrounding the target words to be considered as occurring together).

The existing works focusing on structured DSMs are mostly based on dependency relation between words. The basis elements of the context are generally assumed to be tuples (r, w) where w is a word occurring in relation type r with a target word t. The relations typically reflect argument structure (such as subject, object, indirect object) or modification (such as adjective–noun, noun–noun) (Padó & Lapata, 2007).

The idea of exploiting CCG category within structured DSMs approach is motivated by the assumption that CCG category, being a syntactic category, not only provides the syntactic information of a word but also its intrinsic type within semantics. It presumably gives richer knowledge about a word compared with POS tag, and therefore better defines the linguistic context.

As an illustration, consider a verb *fly* occurring in these two simple sentences:

(1) He flies an airplane.

(2) The bird is flying.

Using POS annotation, the word *fly* will be annotated either as *3rd person singular present verb (VBZ)* in (1) or *present participle verb (VBG)* in (2). Meanwhile, parsing the two sentences with CCG will result in the verb *fly* having CCG category *(S\NP)/NP)* in (1) indicating a transitive verb, and *S\NP* in (2) which indicates an intransitive verb.

	fly_VBZ	*fly_VBG*
airplane	1	0
bird	0	1

	fly_(S\NP)/NP)	*fly_S\NP*
airplane	1	0
bird	0	1

Figure 2.5: Co-occurrence matrices with POS tag- and CCG category-context

	fly_VBZ	*fly_VBG*
airplane	1	0
bird	0	1
helicopter	0	1
bee	1	0

	fly_(S\NP)/NP)	*fly_S\NP*
airplane	1	0
bird	0	1
helicopter	1	0
bee	0	1

Figure 2.6: Co-occurrence matrices with POS tag- and CCG category-context, with additional target words

Using these syntactic information, the co-occurrence matrices for target words *airplane* and *bird* can then be constructed as shown in Figure 2.5. Now if we have two new sentences:

(3) John is flying a helicopter.

(4) The bee flies away.

the co-occurrence matrices containing also the new target words *helicopter* and *bee* will be as shown in Figure 2.6. We could see from the first matrix that the POS tags basically only give distinction on the verb *fly* morphologically, thus in this case fail to discriminate target words *airplane* and *helicopter* from *bird* and *bee*.

On the contrary, the target words in the second matrix are naturally grouped into two semantic classes [*airplane*, *helicopter*] and [*bird*, *bee*]. This is due to the ability of CCG category to distinguish words that tend to be the direct object of transitive verbs from words that tend to be the subject of intransitive verbs.

Hence extending linguistic context with CCG category can be viewed as an indirect approach to exploit syntactic relation between words in distributional models, as what is explicitly done in modelling distributional semantics based on dependency grammar.

2.3.2 Building the Co-occurrence Matrix

At an abstract level, building the co-occurrence matrix is a simple process of counting events, where a single event is the occurrence of a certain item in a certain situation. In distributional models, each element in the matrix corresponds to the frequency of word occurrences in a certain linguistic context.

In practice, the process of building the matrix can be complicated especially when the corpus is large. Moreover, to provide the embodiment of syntactic information within the definition of context, a simple tokenized corpus is not enough. Parsing the corpus is a necessary step prior to use it for building the co-occurrence matrix.

Therefore, in the case of including CCG category into the linguistic context, CCG parsed corpus is needed. A designated algorithm is then utilized to extract the CCG categories of words out of the corpus while computing the co-occurrence frequencies, according to the format of the annotated corpus. Sometimes preprocessing of the corpus is needed to record events and their frequencies in a hash table, a database, or a search engine index, and build the co-occurrence matrix accordingly in order to speed up the process.

2.3.3 Weighting the Elements

The idea of weighting is to give more weight to surprising events and less weight to expected events. In information theory, a surprising event has higher information content than an expected event (Shannon, 1948). For example, in measuring the similarity between *cat* and *dog*, the context *fur* is more discriminative of their similarity than the context *have*.

The most popular way to formalize this idea for term–document matrices is the *tf-idf* (term frequency–inverse document frequency) family of weighting functions (Jones, 1972). An element gets a high weight when the corresponding term is frequent in the corresponding document (i.e. *tf* is high), but the term is rare in other documents in the corpus (i.e. *df* is low, and thus *idf* is high). This weighting scheme is very popular in information retrieval tasks.

An alternative to tf-idf is *Pointwise Mutual Information (PMI)* based on the idea of *Mutual Information* by Church and Hanks (1989), which is more suitable for word–context matrices. Let F be a word–context frequency matrix with n_r rows and n_c columns. PMI for an element in the i-th row and j-th column of F (PMI_{ij}) is defined as follows:

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{n_r}\sum_{j=1}^{n_c} f_{ij}} \tag{2.13}$$

$$p_{i} = \frac{\sum_{j=1}^{n_c} f_{ij}}{\sum_{i=1}^{n_r}\sum_{j=1}^{n_c} f_{ij}} \tag{2.14}$$

$$p_j = \frac{\sum_{i=1}^{n_r} f_{ij}}{\sum_{i=1}^{n_r}\sum_{j=1}^{n_c} f_{ij}} \tag{2.15}$$

$$\text{PMI}_{ij} = \log \frac{p_{ij}}{p_i \cdot p_j} \tag{2.16}$$

where f_{ij} is the frequency of word w_i co-occurs with the context c_j. Hence p_{ij} is the estimated probability that the word w_i co-occurs with the context c_j, p_i is the estimated probability of the word w_i, and p_j is the estimated probability of the context c_j.

If word w_i and context c_j are statistically independent, the probability that they occur together is given by the product $p_i \cdot p_j$. Hence the ratio between p_{ij} and $p_i \cdot p_j$ measures the degree of statistical dependence between word w_i and context c_j. Since the frequency of context c_j functions as the denominator in the ratio, PMI tends to favour less frequent context, provided that it has higher information content than contexts that occur together with a lot of words.

The log of the ratio corresponds to a form of correlation, which is positive if there is an interesting semantic relation between w_i and c_j, and negative otherwise. A variation of PMI is Exponential PMI (EPMI) which is the exponential function of PMI, resulting in the equation that only measures the ratio.

$$\text{EPMI}_{ij} = \frac{p_{ij}}{p_i \cdot p_j} \tag{2.17}$$

Another variation is Positive PMI (PPMI), in which all PMI values that are less than zero are replaced with zero (Niwa & Nitta, 1994).

$$\text{PPMI}_{ij} = \begin{cases} PMI_{ij} & \text{if } PMI_{ij} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.18}$$

The problem of PMI is its bias towards overestimating the significance of low frequency events (*i.e.*, rare events), so another approach is taken to take the observed frequency of co-occurrence into account. This variant of PMI is called *Local Mutual Information (LMI)* (Evert, 2004) which is an approximation to the log-likelihood ratio (LLR) measure (Dunning, 1993), and is quite similar to raw frequency.

$$\text{LMI}_{ij} = f_{ij} \log \frac{p_{ij}}{p_i \cdot p_j} \tag{2.19}$$

Similar with PMI, there is also a variation of LMI called Positive LMI (PLMI) which replaces all the negative values in the matrix with zero.

$$\text{PLMI}_{ij} = \begin{cases} LMI_{ij} & \text{if } LMI_{ij} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.20}$$

Chapter 3

CCG Parsed Corpus

3.1 Wikipedia & ukWaC

The corpus that will be parsed using CCG and then used to build the distributional models is a concatenation of two considerably large corpora: Wikipedia and ukWaC. Wikipedia corpus is built by extracting the content of Wikipedia English articles. The corpus contains approximately 820 million words put together into 43.7 million sentences.

While ukWaC (Ferraresi, Zanchetta, Baroni, & Bernardini, 2008) is a very large (>2 billion words) corpus of British English. It is built by web crawling, contains basic linguistic annotation (part-of-speech tagging and lemmatization) and aims to serve as a general-purpose corpus of English, comparable in terms of document heterogeneity to traditional balanced resources.

Since the aim was to build a corpus of British English, the crawl was limited to the .uk Internet domain. It was created in 2007 as part of the WaCky[1] project, an informal consortium of researchers interested in the exploration of the web as a source of linguistic data.

3.2 CCG Parsing

3.2.1 Evaluating C&C Parser Result

The updated version of C&C parser used to build the CCG parsed corpora was acquired from the author of (Honnibal et al., 2010), along with the new models and supertagger required. Before running the parser in the large corpora, we first inspect the performance of the parser to ensure that the problems addressed previously in section 2.2.3 are successfully eliminated.

The graphical representation of the C&C parser output can be obtained by specifying the output format as XML, and using the available XSLT template the result can then be changed into HTML format to be displayed nicely in the browser. As can be seen in Figure 3.1, the relative pronoun *which* is now treated correctly as an N modifier.

3.2.2 Preprocessing

To run C&C parser, the input file must meet the requirement of the parser input format, which is quite simple: tokenized sentences separated with newline. The corpora mentioned

[1] `http://wacky.sslmit.unibo.it/doku.php?id=corpora`

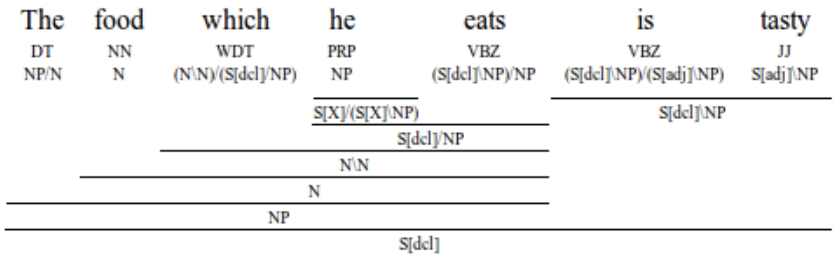


Figure 3.1: CCG derivation for the sentence *The food which he eats is tasty.*

above are actually in the format of MaltParser[2] output, which can be seen in Figure 3.2.

```
<s>
Hooligans   hooligan    NNS    1    4   NMOD
,           ,           ,      2    4   P
unbridled   unbridled   JJ     3    4   NMOD
passion     passion     NN     4    0   ROOT
-           -           :      5    4   P
and         and         CC     6    4   CC
no          no          DT     7    9   NMOD
executive   executive   JJ     8    9   NMOD
boxes       box         NNS    9    4   COORD
.           .           SENT   10   0   ROOT
</s>
```

Figure 3.2: MaltParser output format

Therefore some preprocessing was needed to convert such format into expected input format, by simply getting the first column in each line inside the <s> tag, and put them together into one sentence. This process of converting was done using bash script in Linux.

In the end, by counting the number of lines in the converted file, we get that the number of sentences to be parsed in Wikipedia corpus is 43,745,896, almost half a number of sentences in ukWaC corpus which is 88,214,600. Therefore, the total number of sentences to be parsed from both corpora is approximately 131 million sentences.

3.2.3 Running C&C Parser

To run the parser, one should execute the command:

```
./bin/candc --models models --candc-super super --input sample --output sample.out
--candc-printer xml
```

with options:

```
--models <arg>:  the candc model(s) directory (REQUIRED)
--candc-super <arg>:  the super model(s) directory
--input <arg>:  the input file to read from
--output <arg>:  the output file to write to
--candc-printer <arg>:  parser printing style [deps, grs, prolog, boxer, ccgbank, xml, debug,
js]
```

Because the size of the corpora used is very large, some strategies were needed to do the parsing process efficiently. Each corpus was split into chunk files with 1 million sentences

[2] http://www.maltparser.org/

```
<candc>
<ccg>
 <rule type="ba" cat="NP">
  <rule type="lex" cat="NP">
   <lf start="0" span="1" word="Hooligans" lemma="hooligan" pos="NNS" chunk="I-NP" entity="O" cat="N" />
  </rule>
  <rule type="rp" cat="NP\NP">
   <rule type="lp" cat="NP\NP">
    <lf start="1" span="1" word="," lemma="," pos="," chunk="O" entity="O" cat="," />
    <rule type="ba" cat="NP">
     <rule type="lex" cat="NP">
      <rule type="fa" cat="N">
       <lf start="2" span="1" word="unbridled" lemma="unbridled" pos="JJ" chunk="I-NP" entity="O" cat="N/N" />
       <rule type="rp" cat="N">
        <lf start="3" span="1" word="passion" lemma="passion" pos="NN" chunk="I-NP" entity="O" cat="N" />
        <lf start="4" span="1" word="-" lemma="-" pos=":" chunk="O" entity="O" cat=":" />
       </rule>
      </rule>
     </rule>
     <rule type="conj" cat="NP\NP">
      <lf start="5" span="1" word="and" lemma="and" pos="CC" chunk="O" entity="O" cat="conj" />
      <rule type="fa" cat="NP">
       <lf start="6" span="1" word="no" lemma="no" pos="DT" chunk="I-NP" entity="O" cat="NP/N" />
       <rule type="fa" cat="N">
        <lf start="7" span="1" word="executive" lemma="executive" pos="NN" chunk="I-NP" entity="O" cat="N/N" />
        <lf start="8" span="1" word="boxes" lemma="box" pos="NNS" chunk="I-NP" entity="O" cat="N" />
       </rule>
      </rule>
     </rule>
    </rule>
   </rule>
   <lf start="9" span="1" word="." lemma="." pos="." chunk="O" entity="O" cat="." />
  </rule>
 </rule>
</ccg>
</candc>
```

Figure 3.3: C&C parser XML output format

each. Then we ran the parser in computer cluster, by submitting the job to the cluster nodes, using command `qsub`.

This way, the parsing process could be done simultaneously, processing up to 20 chunk files at the same time. We limited the number of parallel jobs to make sure that the cluster nodes will not be overloaded, resulting in longer running time.

C&C parser provides several types of output format, but only the prolog and XML formats provide the CCG derivations. We decided to go with XML as the output format since it is more compact and easier to be parsed.

The example of the XML output format is shown in Figure 3.3, depicting the CCG derivation for *"Hooligans , unbridled passion - and no executive boxes ."*. The output is always within <`candc`> tag, while the <`ccg`> tag indicates a sentence. The <`lf`> tag indicates a *leaf* in the derivation, which is the word, together with its attributes such as part-of-speech (POS) and lexical category. And finally, the <`rule`> tag specifies the CCG rule that applies to its children nodes, including the resulting category of the rule application.

3.2.4 Parsed Corpora Statistics

Parsed Sentences

In the Wikipedia corpus, 93.57% sentences were parsed, leaving 2,811,956 sentences not parsed out of 43.7 million sentences. Whereas in the ukWaC corpus, the percentage of parsed sentences is a bit higher, 94.03% leaving 5,267,025 sentences not parsed out of 88.2 million sentences. Finally, in total there are 123.8 million sentences contained in the parsed corpora.

Running Time

The average running time of the parser in the Wikipedia corpus is approximately 12 hours for processing 1 million sentences, 42 milliseconds per sentence. While the average running time in the ukWaC corpus is around 16 hours per 1 million sentences, 57 milliseconds per sentence.

This is most probably because the sentences in ukWaC corpus are mostly longer compared with sentences in Wikipedia corpus. From the statistics we found that the average sentence length in ukWaC corpus is 23.08 words, while it is only 21.34 words in Wikipedia corpus.

CCG Rules

Total number of rule types found in both corpus is 15. The type of the rules and their frequencies are presented in Table 3.1. Most of them are combinatory rules which are already mentioned in section 2.2 (with new abbreviation terms for each rule), plus a few additional rules.

Rule type		Wikipedia	ukWaC
fa	forward application ($>$)	512,142,749	1,149,632,154
ba	backward application ($<$)	173,928,884	384,028,684
fc	forward composition ($>B$)	7,716,183	25,220,808
bc	backward composition ($<B$)	323,444	1,317,728
bx	backward crossed composition ($<B_x$)	8,839,360	27,426,570
gbc	generalized backward composition ($<B^n$)	18	170
gbx	generalized backward crossed composition ($<B^n_x$)	169,410	556,820
tr	type-raising ($>T$)	2,397,114	10,587,861
conj	coordination (coord.)	25,481,332	60,148,246
lex	type-changing	129,907,210	270,427,004
funny	special coordination	343,751	950,773
lp	left punctuation	29,060,928	46,105,108
rp	right punctuation	73,119,766	131,934,943
ltc	left punctuation type-changing	881,583	1,908,266
rtc	right punctuation type-changing	744,904	2,160,063

Table 3.1: CCG rule types and frequencies

Notice that there is no instance of rule type: fx – *forward crossed composition* ($>B_x$), gfc – *generalized forward composition* ($>B^n$) and gfx – *generalized forward crossed composition* ($>B^n_x$) found in both corpora.

The *lex* rule type contains all unary type-changing rules. The most common rule is the one that changes N into NP. Other than that there are several rules of which the complete list is provided in (Bos et al., 2004).

The *funny* rule is introduced to accommodate some coordination in the original Penn Treebank which were difficult to convert into CCGbank. It is a special form of coordination rule which is defined as follows:

$$conj \quad N \Rightarrow N \tag{3.1}$$

Both *lp* and *rp* rule types contain punctuation rules, which deal with: comma, colon, semicolon, period and round bracket. The *lp* rules absorb punctuations to the left, while *rp* rules absorb punctuations to the right. Both rules for different punctuations may be applied to different set of categories.

Both *ltc* and *rtc* rule types contain some binary type-changing rules involving commas, for example:

$$, \quad NP \Rightarrow (S\backslash NP)\backslash(S\backslash NP) \tag{3.2}$$

$$NP \quad ,\Rightarrow S/S \tag{3.3}$$

Rule type	Wikipedia	ukWaC
gbc	16	144
gbx	149,207	494,766
tr	1,812,694	7,921,815
conj	18,124,354	40,250,980
lex	38,676,331	74,143,940
funny	340,012	938,458
lp	15,513,367	27,247,397
rp	40,651,183	81,243,989
ltc	861,193	1,860,917
rtc	717,251	2,056,868

Table 3.2: CCG rule types and number of sentences using them

Rule type combination	Wikipedia	ukWaC
fa	2,993,255	2,642,287
ba	233,263	391,308
fc	1,979	11,513
bc	8	548
bx	156	8,084
fa-ba	25,177,096	48,384,119
fa-ba-fc	2,567,966	4,797,034
fa-ba-bc	23,278	76,961
fa-ba-bx	5,239,143	13,690,052
fa-ba-fc-bc	3,231	11,097
fa-ba-fc-bx	862,346	2,362,429
fa-ba-bc-bx	12,069	45,147
fa-ba-fc-bc-bx	3,124	12,145
fa-tr	2	6
ba-tr	3	4
fc-tr	2,374	10,912
bc-tr	338	3,713
bx-tr	0	0
fa-ba-tr	321	336
fa-ba-fc-tr	513,616	2,215,358
fa-ba-bc-tr	140,058	645,988
fa-ba-bx-tr	69,960	237,604
fa-ba-fc-bc-tr	22,960	99,379
fa-ba-fc-bx-tr	27,330	96,719
fa-ba-bc-bx-tr	21,043	85,459
fa-ba-fc-bc-bx-tr	14,422	84,203

Table 3.3: Combination of CCG rule types and number of sentences using them

Table 3.2 illustrates the number of sentences in the corpora which used the rules listed. Whereas Table 3.3 gives the number of sentences in which the various combination of *fa*, *ba*, *fc*, *bc* and *bx* rules occur. Those sentences could also contain any of the rules in Table 3.2 but the *tr* rule.

From Table 3.3 it is shown that sentences with only rule type *fa* and *ba* applied are the most frequent, around 61.58% of Wikipedia corpus and 58.33% of ukWaC corpus. While the total number of sentences which used the composition rules (fc, bc, bx, gbc, gbx) are 21.65% of Wikipedia corpus and 25.93% of ukWaC corpus. From these figures one could infer that generally sentences in ukWaC corpus are more complex than sentences in Wikipedia corpus.

POS tags		Wikipedia	ukWaC
CC	coordinating conjunction	26,065,946	63,774,967
CD	cardinal number	27,582,664	44,814,363
DT	determiner	83,866,291	177,276,713
EX	existential *there*	1,026,704	3,481,138
FW	foreign word	731,391	456,393
IN	preposition	104,659,572	218,938,435
JJ	adjective	53,606,895	125,629,192
JJR	adjective, comparative	1,648,038	5,065,425
JJS	adjective, superlative	1,494,236	3,218,599
LS	list item marker	6,627	20,975
MD	modal	3,410,225	22,967,183
NN	noun, singular or mass	113,952,845	270,467,779
NNS	noun, plural	41,717,255	108,889,450
NNP	proper noun, singular	121,165,225	186,528,770
NNPS	proper noun, plural	2,902,874	3,778,922
PDT	predeterminer	170,580	955,786
POS	possessive ending	5,177,740	13,466,773
PRP	personal pronoun	14,457,430	54,093,218
PRP$	possessive pronoun	8,518,936	21,037,041
RB	adverb	22,601,687	64,238,163
RBR	adverb, comparative	618,129	2,273,895
RBS	adverb, superlative	498,054	996,753
RP	particle	1,402,169	4,981,071
SYM	symbol	26,121	160,148
TO	*to*	16,023,870	46,776,082
UH	interjection	67,440	262,357
VB	verb, base form	13,156,420	64,421,953
VBD	verb, past tense	31,188,973	36,978,754
VBG	verb, gerund or present participle	11,642,056	32,344,771
VBN	verb, past participle	23,225,156	45,340,259
VBP	verb, non-3rd person singular present	6,599,050	34,471,926
VBZ	verb, 3rd person singular present	18,126,460	44,182,017
WDT	wh-determiner	3,618,350	8,210,267
WP	wh-pronoun	1,840,866	5,977,168
WP$	possessive wh-pronoun	109,094	203,978
WRB	wh-adverb	2,050,289	6,553,588

Table 3.4: Penn Treebank part-of-speech (POS) tags and frequencies

Part-of-Speech (POS) Tag

The total number or POS tags found in the corpora is 49, where 36 of them are the POS tags from the Penn Treebank, and the rest of them are introduced in the CCGbank. Table 3.4 shows the list of Penn Treebank POS tags together with the number of occurrences in each corpus, while Table 3.5 contains the additional POS tags from the CCGbank along with their frequencies.

Looking at Table 3.5 we can conclude that the tagger employed by the parser still performs quite poorly in dealing with quotes, since the number of occurrences of LQU and RQU tags are asymmetric.

Categories

There are 35 atomic categories out of 728 total categories found, which is shown in Table 3.6. N category appears as the most frequent followed by NP category. The significant difference

POS tags		Wikipedia	ukWaC
#		3,362	33,646
$		3,707	196,730
,		42,077,301	70,991,281
.		40,363,743	81,886,490
:		929,621	6,348,086
;		1,022,069	2,856,125
AS	*as*	15,296	46,286
LQU	left quote	7,937,551	6,036,960
LRB	left round bracket	6,144,064	10,225,392
NP	*neither*	57	104
RQU	right quote	3,960,300	1,409,591
RRB	right round bracket	6,181,773	10,576,211
SO	*so, too*	91,750	486,574

Table 3.5: CCGbank additional part-of-speech (POS) tags and frequencies

in number of occurrences of the pair LQU-RQU categories and LRB-RRB categories shows that the parser still has problem in dealing with quote and round bracket punctuations.

Atomic category	Wikipedia	ukWaC
,	42,077,648	70,992,742
.	40,363,743	81,886,490
:	869,970	6,100,900
;	1,022,069	2,856,125
LQU	7,937,551	6,036,960
LRB	1,476,911	3,360,696
N	366,257,375	753,336,966
NP	276,812,526	573,112,694
NP[expl]	544,979	1,435,013
NP[thr]	1,010,454	3,492,848
N[num]	1,647,551	1,857,655
PP	55,398,937	124,070,147
PR	1,085,505	3,782,118
RQU	3,960,300	1,409,591
RRB	6,181,773	10,576,211
S	22,298	135,652
S[X]	26,930	124,431
S[adj]	10,518	115,890
S[b]	443,065	2,485,920
S[bem]	12,473	18,800
S[dcl]	65,780,154	147,104,069
S[em]	2,247,658	7,791,495
S[for]	23,352	129,271
S[frg]	3,876	56,876
S[intj]	2,623	69,263
S[inv]	22,357	52,644
S[ng]	184,291	811,878
S[poss]	12,795	100,463
S[pss]	328,112	826,139
S[pt]	107,771	192,774
S[q]	61,326	1,198,883
S[qem]	339,052	2,553,709
S[to]	13,812	264,261
S[wq]	75,681	1,536,805
conj	26,471,297	62,154,608

Table 3.6: Atomic categories and frequencies

Chapter 4

Evaluation Task

4.1 Word Categorization

Categorization tasks play an important role in cognitive research on concepts and meaning, since they act as tools to arrange concepts hierarchically into taxonomies (Baroni & Lenci, 2010). Research in distributional semantic approaches has always been interested in the possibility to use distributional similarity to group words which are semantically similar together. Categorization requires not only the models' ability to detect synonyms but also less strictly related words with the same hypernym.

We will focus on concrete noun categorization and verb categorization, which will be perceived as a clustering task. The dataset that will be used is taken from shared tasks published in ESSLLI 2008 Workshop.[1]

4.1.1 Concrete Noun Categorization

The dataset consists of 44 concrete nouns grouped into 6 semantic categories, 4 categories of natural objects and 2 categories of man-made artifacts. Since the dataset is organized hierarchically, there will be three clustering experiments, varying the number of classes and consequently the degree of generality:

- 6-way clustering, models will be tested on their ability to categorize the nouns into most fine-grained classes of the dataset: *bird* ("peacock"), *groundAnimal* ("lion"), *fruitTree* ("cherry"), *green* ("potato"), *tool* ("hammer"), *vehicle* ("car").
- 3-way clustering, models will be tested on their ability to categorize the nouns into 3 classes: *animal* (superclass of *bird* and *groundAnimal*), *vegetable* (superclass of *fruitTree* and *green*), and *artifact* (superclass of *tool* and *vehicle*).
- 2-way clustering, models will be tested on their ability to categorize the nouns into the two top classes: *natural* (superclass of *animal* and *vegetable*) and *artifact*.

Table 4.1 shows the number of concrete nouns contained in each semantic class for each clustering task.

4.1.2 Verb Categorization

The dataset consists of 45 verbs which belong to 9 semantic classes. Similar with the noun categorization task, there will be two experiments:

[1] http://wordspace.collocations.de/doku.php/data:esslli2008:start

6-way clustering		3-way clustering		2-way clustering	
class	number	class	number	class	number
bird	7	animal	15	natural	24
groundAnimal	8				
fruitTree	4	vegetable	9		
green	5				
tool	13	artifact	20	artifact	20
vehicle	7				

Table 4.1: Distribution of concrete nouns in each semantic class

- 9-way clustering, models will be tested on their ability to categorize the verbs into the most fine-grained classes of the dataset: *communication* ("talk"), *mentalState* ("know"), *motionManner* ("run"), *motionDirection* ("arrive"), *changeLocation* ("carry"), *bodySense* ("smell"), *bodyAction* ("eat"), *exchange* ("buy"), *changeState* ("destroy").
- 5-way clustering, models will be tested on their ability to categorize the verbs into 5 classes: *cognition* (superclass of *communication* and *mentalState*), *motion* (superclass of *motionManner*, *motionDirection*, *changeLocation*), *body* (superclass of *bodySense* and *bodyAction*), *exchange*, and *changeState*.

9-way clustering		5-way clustering	
class	number	class	number
communication	5	cognition	10
mentalState	5		
motionManner	5	motion	15
motionDirection	5		
changeLocation	5		
bodySense	5	body	10
bodyAction	5		
exchange	5	exchange	5
changeState	5	changeState	5

Table 4.2: Distribution of verbs in each semantic class

Table 4.2 shows the number of verbs contained in each semantic class for each clustering task.

4.2 Clustering Tools: CLUTO

Clustering is the task of assigning a set of objects into classes (*i.e.*, clusters) so that the objects in the same cluster are more similar (in some sense or another) to each other than to those in other clusters. In machine learning, clustering is often called an *unsupervised learning* task since no class values denoting the grouping of the objects are given prior to clustering, as opposed to *supervised learning*.

One of the clustering tools which is widely used is CLUTO toolkit (Karypis, 2003). CLUTO is a software package for clustering low- and high-dimensional datasets and for analyzing the characteristics of the various clusters. CLUTO is well-suited for clustering data sets arising in many diverse application areas including information retrieval, customer purchasing transactions, web, GIS, science, and biology.

CLUTO's distribution is available online[2], which consists of both stand-alone programs

[2] `http://glaros.dtc.umn.edu/gkhome/cluto/cluto/overview`

and a library via which an application program can access directly the various clustering and analysis algorithms implemented in CLUTO.

4.2.1 Clustering Algorithm

There are three different kinds of clustering algorithm provided by CLUTO, which are based on the agglomerative, partitional, and graph-partitioning paradigms.

Agglomerative Clustering

The agglomerative clustering algorithm approaches the clustering problem with a bottom-up perspective. The algorithm begins by placing each object in its own cluster. Next, a series of iterations are performed that successively merges pairs of these clusters into larger clusters. This type of clustering is called hierarchical clustering, since the clusters produced are in hierarchical structure.

The merge order is guided by a criterion function that determines the most advantageous merge in each iteration, by considering the similarities between clusters. The algorithm stops when either all objects have been merged into one cluster or a specified number of clusters is obtained.

Partitional Clustering

CLUTO provides two partitional algorithms: *direct* and *repeated-bisection*. The direct partitional algorithm performs a k-way clustering by first randomly choosing k objects as the initial centroids (*i.e.*, seeds) of each cluster. The remaining objects are then successively added to the cluster that best optimizes the criterion function. After all objects have been assigned to a cluster, random objects are chosen for re-evaluation and are reassigned to a different cluster if it better optimizes the criterion function. The algorithm concludes after a specified number of re-evaluation have occurred.

The other partitional algorithm, repeated-bisection, works similarly to direct algorithm. However only two initial seeds are chosen, partitioning all objects into two clusters. Then one of the two clusters is selected and it is further bisected, leading to a total of three clusters. The process of selecting and bisecting a particular cluster continues until k clusters are obtained. Each of these bisections is performed so that the resulting two-way clustering solution optimizes a particular criterion function.

Graph-Partitional Clustering

The last algorithm supported by CLUTO is graph-partitioning-based clustering. Graph partitioning finds clusters with different characteristics than those found by the previous algorithms. The algorithm treats clustering as a graph partitioning problem by constructing a sparse graph, where the objects are represented as vertices and the edges connect objects whose similarity is above a given threshold. The partitioning is performed using a highly efficient multilevel graph-partitioning algorithm.

4.2.2 Cluster Analyzer

CLUTO also provides tools for analyzing the discovered clusters to understand the relations between the objects assigned to each cluster and the relations between the different clusters, and tools for visualizing the discovered clustering solutions. CLUTO can identify the features that best describe and/or discriminate each cluster.

The set of *descriptive* features is determined by selecting the columns that contribute the most to the average similarity between the objects of each cluster. On the other hand, the

set of *discriminating* features is determined by selecting the columns that are more common in the cluster compared to the rest of the objects (Karypis, 2003). These set of features can be used to gain a better understanding of the set of objects assigned to each cluster and to provide brief summaries about the cluster's contents.

4.2.3 Clustering Quality Measure

To evaluate the cluster quality, there are two standard measures which are available in CLUTO: *entropy* and *purity* (Zhao & Karypis, 2001).

Entropy

Entropy is sometimes referred to as a measure of the amount of "disorder" in a system. In information theory, entropy is a measure of the uncertainty associated with a random variable, and usually refers to the Shannon entropy (Shannon, 1948) defined as follows:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log p(x_i) \tag{4.1}$$

In the context of measuring clustering quality, entropy can be used to measure how disorder a cluster is by analyzing how the various semantic classes are distributed in the cluster. Higher value of entropy indicates higher degree of disorder, hence worse quality of a cluster.

The entropy measure is defined based on Shannon entropy by associating $p(x_i)$ with the proportion of each semantic class in the cluster. Given a particular cluster S_r of size n_r, and n_r^i as the number of objects of the ith class that were assigned to the rth cluster, the proportion of the ith semantic class in the cluster is n_r^i/n_r.

The entropy of a cluster is maximum if elements of all classes present in the same proportion. Consider that $n_r^i = 1$ for $i = 1, ..., q$ (consequently $n_r = q$) where q is the number of classes in the dataset, then the entropy will be:

$$\begin{aligned} H(S_r) &= -\sum_{i=1}^{q} \frac{1}{q} \log \frac{1}{q} \\ &= -\log \frac{1}{q} = -(\log 1 - \log q) = \log q \end{aligned}$$

Since the maximum entropy of a cluster is $\log q$, the entropy measure for a given cluster is then defined as follows:

$$E(S_r) = -\frac{1}{\log q} \sum_{i=1}^{q} \frac{n_r^i}{n_r} \log \frac{n_r^i}{n_r} \tag{4.2}$$

with $\frac{1}{\log q}$ as the normalization factor so that the entropy value is at the interval $[0, 1]$. The entropy of the entire clustering is then the sum of the individual cluster entropies weighted according to the cluster size:

$$Entropy = \sum_{r=1}^{k} \frac{n_r}{n} E(S_r) \tag{4.3}$$

Purity

Another quality measure is purity, which is more straightforward to define the quality of a cluster. It basically measures the proportion of the most frequent class in the cluster. If a cluster only contains objects from one class, then the purity value is 1, denoting the best quality of a cluster. The formal definition of purity of a cluster is as follows:

$$P(S_r) = \frac{1}{n_r} \max_i(n_r^i) \tag{4.4}$$

Using the similar definition as entropy, the purity of the whole clustering solution is then again the weighted sum of the individual cluster purities:

$$Purity = \sum_{r=1}^{k} \frac{n_r}{n} P(S_r) \tag{4.5}$$

Chapter 5

Experiment

5.1 Distributional Models

5.1.1 Target Words and Context Selection

The first steps required in constructing distributional semantic models are to make a list of the target words of interest, and to define the linguistic context including deciding what to take as the basis elements of the context. These process of selecting target words and context will automatically determine the dimensions of the co-occurrence matrix.

The list of target words is extracted from the corpus, and is the composition of:

- 10,000 most frequent nouns (excluding proper names and nouns containing numbers)
- 5,000 most frequent verbs (excluding auxiliary and modal verbs)
- 5,000 most frequent adjectives

Each word in the list is distinguished by its part-of-speech tag, by concatenating it with "_n" for nouns, "_v" for verbs, and "_jj" for adjectives. We make sure that the dataset used in the evaluation tasks, *i.e.*, list of 44 nouns and 45 verbs (See Section 4.1), is included in the list of target words. In the end, the total number of target words is 20,001 with the addition of a noun "chisel" which originally does not appear in 10,000 most frequent nouns.

We aim to make comparison of the performance of CCG-based DSM against enhanced POS-based DSM, thus we build two kind of distributional models, *CCG-DSM* and *POS-DSM*. For each target word, its linguistic context is defined as vector of words annotated with their CCG categories (for CCG-based model) and part-of-speech tag (for POS-based model). The vectors for both models are then extracted from the corpus using these criteria:

- CCG-DSM – 20,000 most frequent CCG categories tagged words.
- POS-DSM – 20,000 most frequent POS tagged words.

The co-occurrence matrices for both models are then automatically characterized as 20,001 $\times$ 20,000 matrices.

In order to investigate the impact of including function words as linguistic context, we do not employ any stop words list in defining the previously explained contexts of DSMs. To evaluate the CCG-based model containing function words in the linguistic context, we build another model based on CCG-DSM, however its linguistic context will only contain content words. We first decide the list of function words, then remove them from the co-occurrence matrix of the model.

```
initialize co-occurrence matrix m
for each line l of the corpus
    if line l contains 'word', indicating a word
        extract the lemma, part-of-speech (pos) and CCG category (ccg) of word w
        if pos is noun or verb or adjective
            if lemma_pos is in target words list
                mark word w as target word
        if lemma_ccg (for CCG-DSM) or lemma_pos (for POS-DSM) is in context vector
            mark word w as context feature
        if word w is marked as target word or context feature (or both)
            insert word w into vector v
    else if line l contains '</ccg>', indicating the end of sentence
        for each word w in vector v
            if word w is target word
                target ← lemma_pos of word w
                for each word z in vector v
                    if word z is context feature
                        context ← lemma_ccg (for CCG-DSM) or
                                  lemma_pos (for POS-DSM) of word z
                        pair target with context and update the value of matrix m
        empty vector v
```

Figure 5.1: Algorithm for building co-occurrence matrix

The words considered as function words are extracted from the corpus with the criterion of having POS tags listed in Table A.1, with the addition of primary verbs *be*, *do*, and *have*. The complete list of extracted function words is available in Table A.2 (see Appendix A.1).

In the end, removing the function words from the modified model results in co-occurrence matrix with different dimension, which is 20,001 × 18,501 matrix.

5.1.2 Building Co-occurrence Matrices

To build the co-occurrence matrices, CCG parsed corpus which previously explained in Chapter 3 is needed. The format of CCG parsed corpus is XML format as illustrated in Figure 3.3, where each sentence is identified by <ccg> tag. Some lines inside the <ccg> tag indicate the words (tagged with syntactic attributes) composing the sentence, while the other lines indicate the CCG rules that combine them together. Among the syntactic attributes that give features to each word, *lemma*, *part-of-speech*, and *CCG category* are the ones that will be exploited upon constructing the co-occurrence matrix.

We define two kinds of algorithm to build the co-occurrence matrix, according to the output format of CCG parsed corpus described above. The first algorithm which is shown in Figure 5.1 is written based on the idea that words are said to co-occur together if they are in the same sentence. The second algorithm presented in Figure 5.2 is a modified version of the first one, with specified number of window size as input. Using the second algorithm, context words are said to co-occur with target words if they appear before and after target words within a certain window size.

To accelerate the process of checking whether a word is contained in target words list or context features list, we employ the use of index based on alphabetic order. So instead of looping through the list of 20,001 target words or 20,000 context features, the algorithm will use the index files to locate the start and end index to search for a matching string.

Using the same strategy as running the C&C parser, we run the algorithm in computer cluster. The CCG parsed corpus used is already split into chunk files with less than 1 million

```
initialize co-occurrence matrix m
for each line l of the corpus
    if line l contains 'word', indicating a word
        extract the lemma, part-of-speech (pos) and CCG category (ccg) of word w
        if pos is noun or verb or adjective
            if lemma_pos is in target words list
                mark word w as target word
        if lemma_ccg (for CCG-DSM) or lemma_pos (for POS-DSM) is in context vector
            mark word w as context feature
        insert word w into vector v
    else if line l contains '</ccg>', indicating the end of sentence
        for each word w in vector v
            if word w is target word
                target ← lemma_pos of word w
                idx ← index of word w in vector v
                s ← context window size
                before ← words in vector v with index [idx-s, ..., idx-1] (idx-s > 0)
                after ← words in vector v with index [idx+1, ..., idx+s] (idx+s < size of v)
                for each word z in vector before
                    if word z is context feature
                        context ← lemma_ccg (for CCG-DSM) or
                                  lemma_pos (for POS-DSM) of word z
                        pair target with context and update the value of matrix m
                for each word z in vector after
                    if word z is context feature
                        context ← lemma_ccg (for CCG-DSM) or
                                  lemma_pos (for POS-DSM) of word z
                        pair target with context and update the value of matrix m
        empty vector v
```

Figure 5.2: Algorithm for building co-occurrence matrix with specified window size

(approximately 940,000) sentences each, after they went through the C&C parser. This way, the construction of co-occurrence matrices could be done simultaneously, processing up to 20 chunk files at the same time.

For each chunk file the output matrix is saved into temporary file. Another simple algorithm is then run to combine all the output matrices into one big matrix for each distributional model.

5.1.3 DSMs with Weighted Matrices

We use COMPOSE toolkit to build co-occurrence matrices with weighted elements. The toolkit requires as input the co-occurrence matrix in the format of three files:

- <*matrix-name*>.rows, containing a list of target words, separated by newline.
- <*matrix-name*>.cols, containing a list of context features, separated by newline.
- <*matrix-name*>.mat, containing the actual co-occurrence counts. The format is space- or tab-delimited: <*string1*> <*string2*> <*co-occurrence-count*>, where *string1* and *string2* represent the elements in rows and columns respectively.

The script to build the weighted matrix is run by executing the command:

```
python build_pipeline.py [options]
```

The options are:

```
-o <dir>:  output directory
-i <dir>:  input directory
-c <string>:  core matrix name e.g.  ccg.core
-p <string>:  peripheral matrix name e.g.  ccg.core (optional)
-w <string>:  comma separated weighting schemes (raw, pmi, ppmi, epmi, or plmi)
-l <file>:  log file (optional)
-b <int>:  number of bits used to store an element of the matrix (either 32 or 64) (optional)
-h:  help
```

We want to evaluate the performance of DSMs with different types of weighting scheme besides the raw co-occurrence frequencies, therefore all of the weighting schemes provided by the toolkit are used.

5.1.4 Evaluating DSMs

We first prepare the distributional models according to the input format expected by CLUTO (see Section 4.2), particularly `vcluster` program which requires an input file that stores the objects to be clustered in a matrix format. Each row of the matrix represent a single object, and its various columns correspond to the dimensions (*i.e.*, features) of the objects (Karypis, 2003).

CLUTO understands two different input matrix formats: the format suitable for *sparse matrices* and the format suitable for storing *dense matrices*. We use the dense matrix format because the output format of COMPOSE toolkit is similar to dense matrix format defined by CLUTO.

A dense matrix M with n rows and m columns is stored in a plain text file that contains $n+1$ lines. The first line contains information about the size of the matrix, in the format of space-delimited: <*number of rows (n)*> <*number of columns (m)*>. The remaining n lines store the values of the m columns for each one of the rows. In particular, each line contains exactly m space-separated floating point values, such that the i-th value corresponds to the i-th column of M.

While converting the matrices into the expected format, we extract sub-matrices containing only the required target words for evaluation tasks (see Section 4.1). These two processes of converting and extracting sub-matrices are done in one pass.

Finally, we run CLUTO on the prepared matrices by executing the command:

```
./vcluster [optional parameter] matrix-file number-of-cluster
```

with optional parameters:

```
-clmethod=rbr
-rclassfile=<arg>: specifies the name of the file that stores the class-labels of the rows
(i.e., the objects to be clustered)
-rlabelfile=<arg>: specifies the name of the file that stores the labels of the rows
-clabelfile=<arg>: specifies the name of the file that stores the labels of the columns
-showfeatures
-nfeatures=<arg>: specifies the number of descriptive and discriminating features to display
for each cluster when the -showfeatures is used
```

We use CLUTO's built-in repeated bisections with global optimization as the clustering method, which is indicated by `-clmethod=rbr` parameter, to make the models comparable with other existing models. As for the other parameters we accept all of CLUTO's default value.

Specifying the `-rclassfile` parameter is important for the purpose of computing the quality of the clustering solution using quality measures explained in Section 4.2.3. The format of the file is the string labels of corresponding classes for each row (*i.e.*, object) in the matrix, separated by newline.

Whereas specifying the `-clabelfile` parameter is needed for reporting purpose, when the `-showfeatures` option is specified. `-showfeatures` parameter instructs `vcluster` to identify the set of features that are most descriptive of each cluster and the set of features that best discriminate each cluster from the rest of the objects. The number of feature to be displayed is decided by `-nfeatures` parameter, with 5 as default value.

The produced clusters for a matrix with n rows are stored in a file consists of n lines with a single number per line. The number corresponds to the cluster number of each object (*i.e.*, row in the matrix), which run from zero to the total number of clusters minus one. The summary of produced clusters including the clustering quality measures is shown in Figure 5.3.

5.2 Results and Analysis

5.2.1 Concrete Noun Categorization

CCG-DSM vs Other Type of DSMs

Table 5.1 exhibits the clustering result for 44 concrete nouns, of both models CCG-DSM and POS-DSM using various weighting schemes, and also other types of DSM. The *global* score is obtained by summing the 3 purity values, then subtracting it by the sum of 3 entropy values.

Generally, models with PMI-based weighting schemes outperform models with raw frequencies as the co-occurrence matrix values. And since LMI-based weighting schemes use raw frequencies as the multiplication factor, the performance is almost similar with raw frequency models although slightly better. The best weighting scheme for this task is achieved by model with PPMI weighting, CCG-DSM-ppmi, with the global score of 0.805.

```
*******************************************************************************
vcluster (CLUTO 2.1.1) Copyright 2001-03, Regents of the University of Minnesota

Matrix Information ------------------------------------------------------------
Name:  ../Matrices/concnouns-ccg.core.pmi, #Rows:  44, #Columns:  20000, #NonZeros:
880000

Options -----------------------------------------------------------------------
CLMethod=RBR, CRfun=I2, SimFun=Cosine, #Clusters:  6
RowModel=None, ColModel=None, GrModel=SY-DIR, NNbrs=40
Colprune=1.00, EdgePrune=-1.00, VtxPrune=-1.00, MinComponent=5
CSType=Best, AggloFrom=0, AggloCRFun=I2, NTrials=10, NIter=10

Solution ----------------------------------------------------------------------

6-way clustering:  [I2=3.18e+01] [44 of 44], Entropy:  0.333, Purity:  0.727
-------------------------------------------------------------------------------
cid Size ISim ISdev ESim ESdev Entpy Purty | frui vehi tool grou bird gree
-------------------------------------------------------------------------------
0 3 +0.599 +0.006 +0.145 +0.065 0.000 1.000 | 0 0 3 0 0 0
1 8 +0.502 +0.068 +0.219 +0.065 0.210 0.875 | 0 7 1 0 0 0
2 9 +0.502 +0.033 +0.256 +0.069 0.592 0.444 | 4 0 2 0 0 3
3 7 +0.560 +0.029 +0.325 +0.028 0.000 1.000 | 0 0 7 0 0 0
4 10 +0.542 +0.037 +0.310 +0.026 0.530 0.600 | 0 0 0 6 2 2
5 7 +0.470 +0.038 +0.247 +0.059 0.334 0.714 | 0 0 0 2 5 0
-------------------------------------------------------------------------------
6-way clustering solution - Descriptive & Discriminating Features...
-------------------------------------------------------------------------------
Cluster 0, Size:  3, ISim:  0.599, ESim:  0.145
Descriptive:  hammer_N 0.5%, knife_N 0.5%, blade_N 0.5%, drill_N 0.3%, screw_N 0.3%
Discriminating:  hammer_N 0.4%, blade_N 0.3%, tool_N 0.2%, knife_N 0.2%, drill_N 0.2%

Cluster 1, Size:  8, ISim:  0.502, ESim:  0.219
Descriptive:  learning_N 0.1%, teaching_N 0.1%, cognitive_N/N 0.1%, practitioner_N 0.1%,
academic_N/N 0.1%
Discriminating:  egg_N 0.1%, bean_N 0.1%, apple_N 0.1%, egg_N/PP 0.1%, chicken_N 0.1%

Cluster 2, Size:  9, ISim:  0.502, ESim:  0.256
Descriptive:  tomato_N 0.3%, apple_N 0.3%, carrot_N 0.3%, onion_N 0.3%, pepper_N 0.3%
Discriminating:  tomato_N 0.4%, apple_N 0.3%, onion_N 0.3%, pepper_N 0.3%, juice_N 0.3%

Cluster 3, Size:  7, ISim:  0.560, ESim:  0.325
Descriptive:  policy_N/N 0.1%, planning_N/N 0.1%, pencil_N 0.1%, administrative_N/N 0.1%,
implementation_N/PP 0.1%
Discriminating:  species_N/PP 0.2%, wild_N/N 0.1%, eat_S[pss]\NP 0.1%, pencil_N 0.1%,
ink_N 0.1%

Cluster 4, Size:  10, ISim:  0.542, ESim:  0.310
Descriptive:  goat_N 0.1%, pig_N 0.1%, chicken_N 0.1%, administrative_N/N 0.1%, employer_N
0.1%
Discriminating:  disease_N 0.1%, sheep_N 0.1%, cattle_N 0.1%, genetic_N/N 0.1%, animal_N/N
0.1%

Cluster 5, Size:  7, ISim:  0.470, ESim:  0.247
Descriptive:  owl_N 0.2%, species_N/PP 0.2%, duck_N 0.2%, turtle_N 0.2%, bird_N/PP 0.2%
Discriminating:  species_N/PP 0.3%, owl_N 0.3%, bird_N/PP 0.2%, prey_N 0.2%,
endangered_N/N 0.2%

Timing Information ------------------------------------------------------------
I/O: 0.521 sec
Clustering:  1.003 sec
Reporting:  0.238 sec
*******************************************************************************
```

Figure 5.3: Output of `vcluster` for a 6-way clustering

Model	6-way		3-way		2-way		Global
	Entropy	Purity	Entropy	Purity	Entropy	Purity	
Van de Cruys (dependency)[1]	0.173	0.841	0.000	1.000	0.000	1.000	2.668
CCG-DSM-raw	0.782	0.386	0.838	0.500	0.896	0.682	-0.948
CCG-DSM-pmi	0.333	0.727	0.783	0.523	0.911	0.659	-0.118
CCG-DSM-epmi	0.279	0.727	0.213	0.909	0.954	0.591	0.781
CCG-DSM-ppmi	0.269	0.727	0.265	0.864	0.911	0.659	0.805
CCG-DSM-plmi	0.548	0.500	0.838	0.500	0.973	0.591	-0.768
POS-DSM-raw	0.753	0.409	0.838	0.500	0.973	0.591	-1.064
POS-DSM-pmi	0.364	0.659	0.788	0.523	0.911	0.659	-0.222
POS-DSM-epmi	0.279	0.727	0.241	0.886	0.972	0.568	0.689
POS-DSM-ppmi	0.269	0.727	0.265	0.864	0.911	0.659	0.805
POS-DSM-plmi	0.741	0.409	0.838	0.500	0.973	0.591	-1.052
Van de Cruys (unstructured)[1]	0.334	0.682	0.539	0.705	0.983	0.545	0.076
Shaoul[1]	0.770	0.409	0.844	0.523	0.931	0.545	-1.068

[1] Model source: ESSLLI 2008 shared task

Table 5.1: Concrete nouns clustering result

Looking at Table 5.1, CCG-DSM-ppmi is clearly so much better compared with unstructured models, Van de Cruys (unstructured) and Shaoul. Van de Cruys uses LSA approach to build the co-occurrence matrix, which contains co-occurrence frequencies of target words together with the most frequent words of the corpus in a context window (paragraphs of the newspaper). In the opposite, Shaoul uses HAL approach with optimized parameters. The original weighting scheme used by HAL is known as linear ramp, which gives the most weight to words that co-occur near the target word. Shaoul proposed different weighting scheme, inverse linear ramp which gives the most weight to words that co-occur farther from the target word, and also different size of context window.

However, CCG-DSM-ppmi is still not as good as models which employ more complex feature as linguistic context such as Van de Cruys (dependency). Van de Cruys makes use of matrix that contains co-occurrence frequencies of nouns by their dependency relations, which is extracted using Alpino parser, a dependency parser for Dutch.

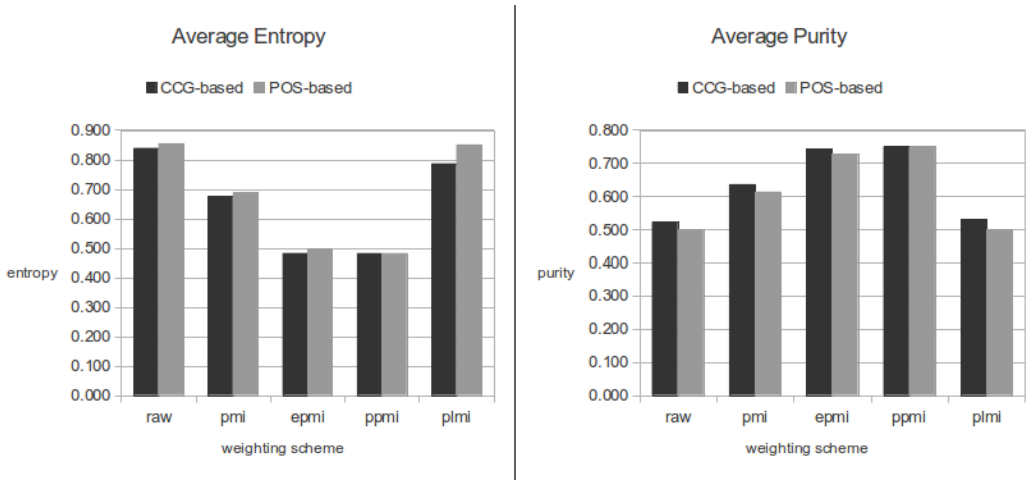


Figure 5.4: Concrete nouns clustering comparison between CCG-DSM and POS-DSM

Better illustration of performance comparison between CCG-DSM and POS-DSM is shown in Figure 5.4. We could see that the average purity and entropy values of CCG-

based models are generally better than the latter, which is expected since CCG categories carry more information than POS tags. Recall that less value of entropy is better.

Various Context Window Size

Model	6-way		3-way		2-way		Global
	Entropy	Purity	Entropy	Purity	Entropy	Purity	
CCG-DSM-raw	0.454	0.614	0.627	0.591	0.991	0.545	-0.322
CCG-DSM-pmi	0.323	0.682	0.349	0.795	0.719	0.795	0.881
CCG-DSM-epmi	0.209	0.750	0.285	0.841	0.957	0.614	0.754
CCG-DSM-ppmi	0.205	0.795	0.381	0.773	0.955	0.614	0.641
CCG-DSM-plmi	0.492	0.523	0.465	0.773	0.954	0.591	-0.024

Table 5.2: Concrete nouns clustering result using modified CCG-DSM with context window of size 2

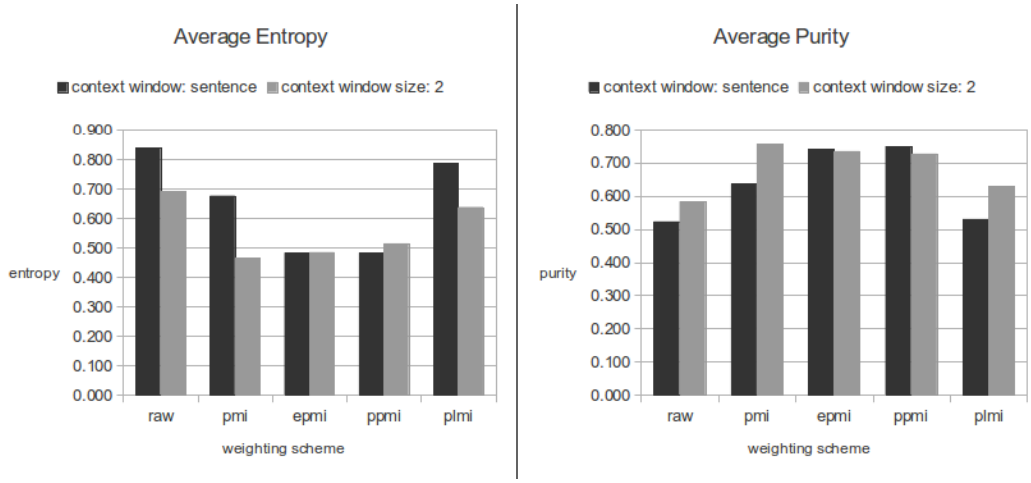


Figure 5.5: Concrete nouns clustering comparison between CCG-DSM using sentence as context window and CCG-DSM using context window of size 2

Table 5.1 presents the performance of CCG-DSM which uses sentence as the context window, and so the window size depends on the length of each sentence. We want to evaluate the effect of smaller context window for the co-occurrence counts extraction. Table 5.2 shows the performance of CCG-DSM which uses context window of size 2.

From the Figure 5.5 which gives better illustration of the comparison between both models with different context window size, we could see that generally the smaller context window size results in significantly better performance, especially for CCG-DSM using raw frequency and PLMI weighting schemes. This might be caused by the existence of irrelevant and misleading context information in larger context window, specifically the grammatical words which have no relation at all with the nouns. Thus explaining why the performance of CCG-DSM with EPMI and PPMI weighting schemes seems not affected by the reduced window size, and even slightly decline.

Function Words Inclusion

Table 5.3 shows the concrete nouns clustering result of modified CCG-DSM which use only content words as context, using various weighting schemes. Figure 5.6 illustrates the

Model	6-way		3-way		2-way		Global
	Entropy	Purity	Entropy	Purity	Entropy	Purity	
CCG-DSM-raw	0.243	0.773	0.067	0.977	0.755	0.682	1.367
CCG-DSM-pmi	0.321	0.705	0.788	0.523	0.911	0.659	-0.133
CCG-DSM-epmi	0.279	0.727	0.213	0.909	0.954	0.591	0.781
CCG-DSM-ppmi	0.241	0.750	0.265	0.864	0.929	0.636	0.815
CCG-DSM-plmi	0.209	0.750	0.265	0.864	0.946	0.614	0.808

Table 5.3: Concrete nouns clustering result using modified CCG-DSM with only content words as linguistic context

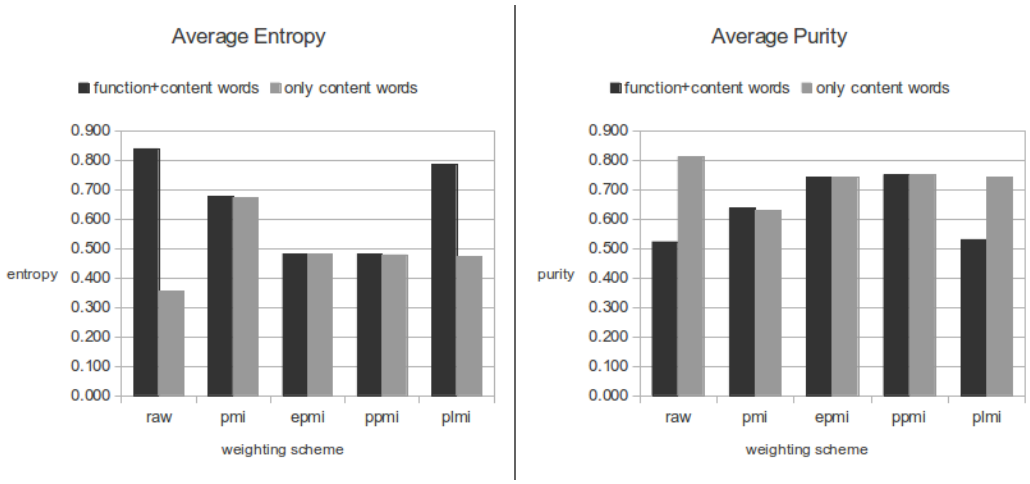


Figure 5.6: Concrete nouns clustering comparison between CCG-DSM with both function and content words, and CCG-DSM with only content words

comparison of CCG-based models which include also function words in the vector of contexts against models which use only content words.

There is almost no difference in performances between both models when using the PMI-based weighting, although the performance of CCG-DSM-ppmi is slightly better with the global score of 0.815. This means that PMI-based models give so little importance to function words due to their high frequencies in the corpus. Therefore, we could conclude that the presence of function words has almost no impact in the models which use PMI-based weighting scheme.

On the contrary, the removal of function words greatly improves the performance of models with raw frequencies in the co-occurrence matrix, as well as models with LMI-based weighting since co-occurrence frequencies play a great role in the equation of the latter weighting scheme.

Qualitative Evaluation

Table 5.4 shows the confusion matrix for each cluster in 6-way clustering, using the CCG-DSM with PPMI weighting. Whereas Table 5.5 gives the top 5 descriptive features that describe each cluster. From both tables we could infer that the model makes unnecessary distinction between *air- & water- vehicles* (cluster 0) and *ground-vehicles* (cluster 2), also between *food-related tools* (cluster 3) and *other tools* (cluster 4). The model however fails to distinguish *fruit* from *green-vegetable* (cluster 1), and also *bird* from *ground-animal* (cluster 5).

Cluster	Classes						Entropy	Purity
	fruit	vehicle	tool	ground[1]	bird	green		
0	0	4	0	0	0	0	0.000	1.000
1	4	0	0	0	1	5	0.526	0.500
2	0	3	1	0	0	0	0.314	0.750
3	0	0	5	0	0	0	0.000	1.000
4	0	0	7	0	0	0	0.000	1.000
5	0	0	0	8	6	0	0.381	0.571

[1] groundAnimal

Table 5.4: Confusion matrix for 6-way concrete nouns clustering

Cluster	Descriptive features
0	**aboard**_((S\NP)\(S\NP))/NP 0.4%, **helicopter**_N 0.4%, **landing**_N/N 0.3%, **patrol**_N/N 0.3%, **missile**_N 0.3%
1	**tomato**_N 0.9%, **carrot**_N 0.8%, **onion**_N 0.7%, **pepper**_N 0.7%, **potato**_N 0.7%
2	**motorcycle**_N 0.5%, **automobile**_N 0.5%, **truck**_N 0.4%, **motorcycle**_N/N 0.4%, **used**_N/N 0.4%
3	**bowl**_N 0.6%, **cup**_N 0.6%, **cup**_N/PP 0.5%, **lid**_N 0.5%, **tea**_N/N 0.5%
4	**knife**_N 0.9%, **pencil**_N 0.8%, **hammer**_N 0.7%, **pen**_N 0.7%, **blade**_N 0.6%
5	**deer**_N 0.6%, **duck**_N 0.5%, **goat**_N 0.5%, **rabbit**_N 0.5%, **lion**_N 0.5%

Table 5.5: Descriptive features for 6-way concrete nouns clustering

Moreover, one noun from *bird* class which is "chicken" is considered to be in the same class with *fruit* and *green-vegetable*, perhaps due to its frequent occurrence in the context of food.

Cluster	Classes			Entropy	Purity
	vegetable	artifact	animal		
0	9	5	1	0.777	0.600
1	0	0	14	0.000	1.000
2	0	15	0	0.000	1.000

Table 5.6: Confusion matrix for 3-way concrete nouns clustering

Cluster	Descriptive features
0	**tomato**_N 0.7%, **onion**_N 0.7%, **pepper**_N 0.6%, **carrot**_N 0.6%, **sauce**_N 0.6%
1	**deer**_N 0.6%, **duck**_N 0.5%, **goat**_N 0.5%, **rabbit**_N 0.5%, **lion**_N 0.5%
2	**knife**_N 0.3%, **blade**_N 0.3%, **arm**_(S[pss]\NP)/PP 0.3%, **pencil**_N 0.3%, **hammer**_N 0.2%

Table 5.7: Descriptive features for 3-way concrete nouns clustering

Similarly, Table 5.6 and Table 5.7 explain the confusion matrix and descriptive features of each cluster in 3-way clustering. The model successfully discriminates nouns of *vegetable* (cluster 0), *animal* (cluster 1), and *artifact* (cluster 2) classes. However, some nouns from the *artifact* class, which are *food-related tools*, are grouped in the same cluster as *vegetable*. Similar with 6-way clustering, one noun from *animal* class which is "chicken" is grouped together with nouns of *vegetable* class.

The problems found in 3-way clustering is also happened in 2-way clustering. Additionally, the model considers 10 nouns of *animal* class to be in the same group as nouns of *artifact* class (cluster 1) instead of *natural* class (cluster 0) as shown in Table 5.8 and Table

5.9, causing the performance to drop significantly for 2-way clustering.

Cluster	Classes		Entropy	Purity
	natural	artifact		
0	14	5	0.831	0.737
1	10	15	0.971	0.600

Table 5.8: Confusion matrix for 2-way concrete nouns clustering

Cluster	Descriptive features
0	**tomato**_N 0.6%, **onion**_N 0.6%, **sauce**_N 0.5%, **carrot**_N 0.5%, **butter**_N 0.5%
1	**tail**_N/PP 0.2%, **toy**_N/N 0.2%, **lion**_N 0.2%, **crane**_N 0.2%, **spot**_(S[dcl]\NP)/NP 0.2%

Table 5.9: Descriptive features for 2-way concrete nouns clustering

Larger Dataset

The AlmuharebPoesio (AP) (Almuhareb & Poesio, 2005) is a balanced dataset containing 402 nouns from 21 WordNet classes, each selected from one of the 21 unique WordNet beginners, and represented by between 13 and 21 nouns (e.g., *ceremony*, *feast*, and *graduation* for the class *social occasion*).

While we extract the target words of interest according to the new dataset, only 303 nouns were found in the co-occurrence matrix of size 20,001 × 20,000. Nevertheless, we repeat the experiment on the available context vector representation of the nouns to confirm whether the same result holds for larger dataset.

We found that the best performance is achieved by CCG-DSM with EPMI weighting scheme, producing the clusters with purity value 0.667 and entropy value 0.307. Compared with the same experiment setting but using the POS-DSM which gives the purity value of 0.650 and entropy value 0.305, we could say that even for larger dataset CCG-DSM still performs better than POS-DSM.

5.2.2 Verb Categorization

CCG-DSM vs Other Type of DSMs

The same experiments as concrete nouns clustering are carried out for verbs clustering. Table 5.10 gives the comparison of models' performance on clustering 45 verbs. Overall, verbs clustering is a harder task compared with nouns clustering, especially for most fine-grained (9-way) clustering illustrated by the average purity that is only around 50%.

Similar with concrete nouns clustering result, PMI-based models perform better than raw frequency or LMI-based weighting scheme. Although the best global score of CCG-DSM with EPMI weighting is less convincing than concrete nouns clustering result, only 0.561, it is still much better than the performance of model that is assumed to be the state-of-the-art, Van de Cruys (dependency) with the global score of 0.351. As explained previously in Section 5.2.1, Van de Cruys employs dependency relations in defining the linguistic context. And naturally, CCG-DSM outperform unstructured models such as Van de Cruys (unstructured) and Shaoul, that are already discussed in Section 5.2.1.

The performance of CCG-DSM-epmi which surpasses the performance of dependency-based DSM gives evidence to the importance of including syntactic categories, in this case CCG categories, within the context vector to define the meaning of verbs.

Model	9-way		5-way		Global
	Entropy	Purity	Entropy	Purity	
Van de Cruys (dependency)[1]	0.408	0.556	0.464	0.667	0.351
CCG-DSM-raw	0.602	0.378	0.790	0.444	-0.570
CCG-DSM-pmi	0.379	0.578	0.476	0.644	0.367
CCG-DSM-epmi	0.364	0.578	0.342	0.689	0.561
CCG-DSM-ppmi	0.408	0.556	0.366	0.689	0.471
CCG-DSM-plmi	0.525	0.444	0.595	0.600	-0.076
POS-DSM-raw	0.600	0.356	0.741	0.444	-0.541
POS-DSM-pmi	0.440	0.489	0.385	0.711	0.375
POS-DSM-epmi	0.401	0.533	0.375	0.667	0.424
POS-DSM-ppmi	0.376	0.578	0.323	0.689	0.568
POS-DSM-plmi	0.551	0.400	0.603	0.578	-0.176
Van de Cruys (unstructured)[1]	0.442	0.556	0.463	0.600	0.251
Shaoul[1]	0.572	0.422	0.755	0.467	-0.438

[1] Model source: ESSLLI 2008 shared task

Table 5.10: Verbs clustering result

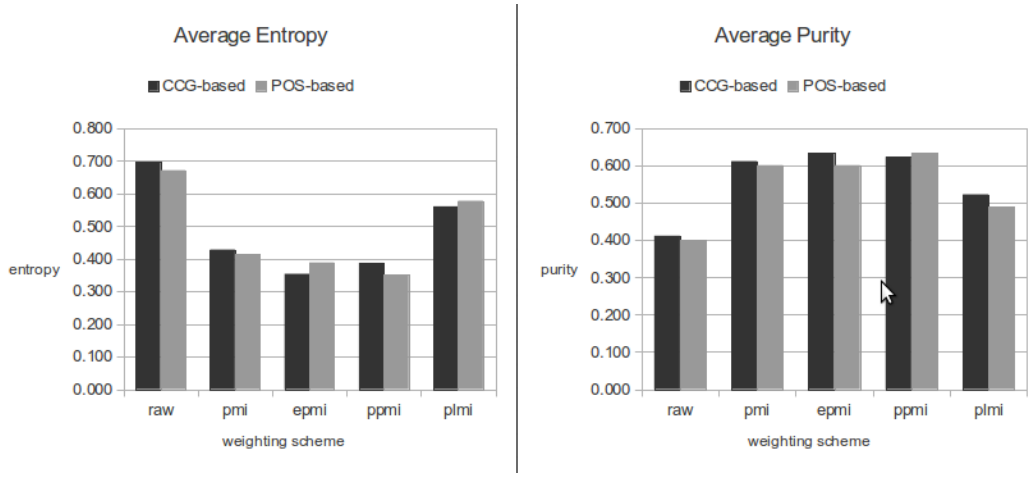


Figure 5.7: Verbs clustering comparison between CCG-DSM and POS-DSM

Looking at Figure 5.7 we could draw a conclusion that basically the average purity values of CCG-DSM on verbs clustering are better than POS-DSM. The same conclusion however cannot be generalized for the overall performance. By observing the global score in 5.10 we could see that some POS-DSM are instead better than CCG-DSM.

Various Context Window Size

Model	9-way		5-way		Global
	Entropy	Purity	Entropy	Purity	
CCG-DSM-raw	0.609	0.378	0.697	0.444	-0.484
CCG-DSM-pmi	0.426	0.533	0.430	0.667	0.344
CCG-DSM-epmi	0.340	0.600	0.305	0.756	0.711
CCG-DSM-ppmi	0.360	0.600	0.363	0.733	0.610
CCG-DSM-plmi	0.555	0.467	0.735	0.467	-0.356

Table 5.11: Verbs clustering result using modified CCG-DSM with context window of size 2

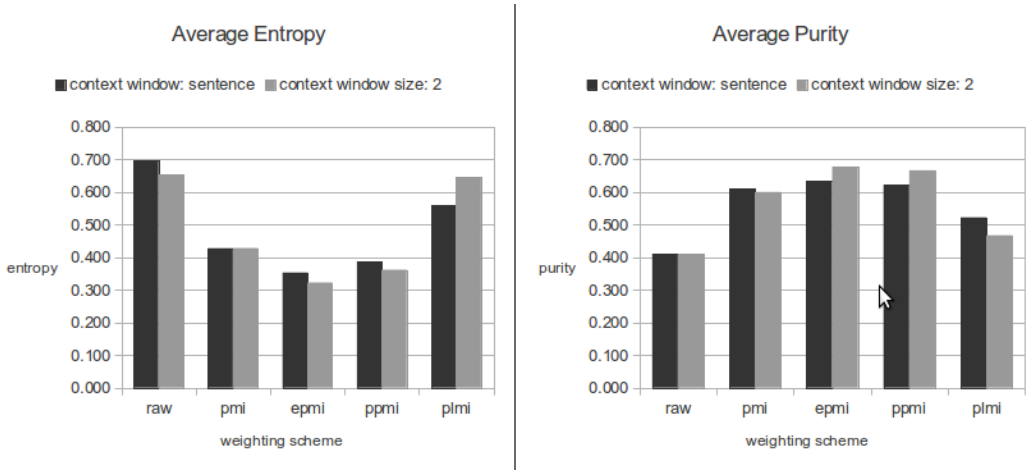


Figure 5.8: Verbs clustering comparison between CCG-DSM using sentence as context window and CCG-DSM using context window of size 2

As in concrete nouns clustering task, Table 5.10 presents the performance of CCG-DSM which uses sentence as the context window. Whereas Table 5.11 shows the performance of CCG-DSM which uses context window of size 2.

Figure 5.8 gives better illustration of the comparison between both models with different context window size. We could see that CCG-DSM using EPMI and PPMI weighting schemes perform better with smaller context window size, the opposite of what happened in concrete nouns clustering.

Looking at the result we might assume that the concept of word meaning between concrete nouns and verbs are different. The meaning of verbs is more closely related to the surrounding words within the closest distance, for example particles, nouns, or adverbs that either follow or precede the verbs. This phenomenon might be related to the nature of verbs in CCG as the type of functions that takes arguments, either the word before or after, to result in another CCG categories.

Function Words Inclusion

Model	9-way		5-way		Global
	Entropy	Purity	Entropy	Purity	
CCG-DSM-raw	0.444	0.556	0.320	0.711	0.503
CCG-DSM-pmi	0.387	0.556	0.453	0.622	0.338
CCG-DSM-epmi	0.364	0.578	0.342	0.689	0.561
CCG-DSM-ppmi	0.390	0.556	0.360	0.711	0.517
CCG-DSM-plmi	0.453	0.489	0.381	0.667	0.322

Table 5.12: Verbs clustering result using modified CCG-DSM with only content words as linguistic context

We want to compare the performance of CCG-DSM with both content and function words, against CCG-DSM with only content words. Table 5.12 exhibits the verbs clustering result of modified CCG-DSM which use only content words as context, using various weighting schemes. Figure 5.9 illustrates the comparison in terms of average values of entropy and average values of purity.

The same phenomenon as in concrete nouns clustering happens, that the performance of model with raw frequencies is drastically improved, hence explaining the improvement of

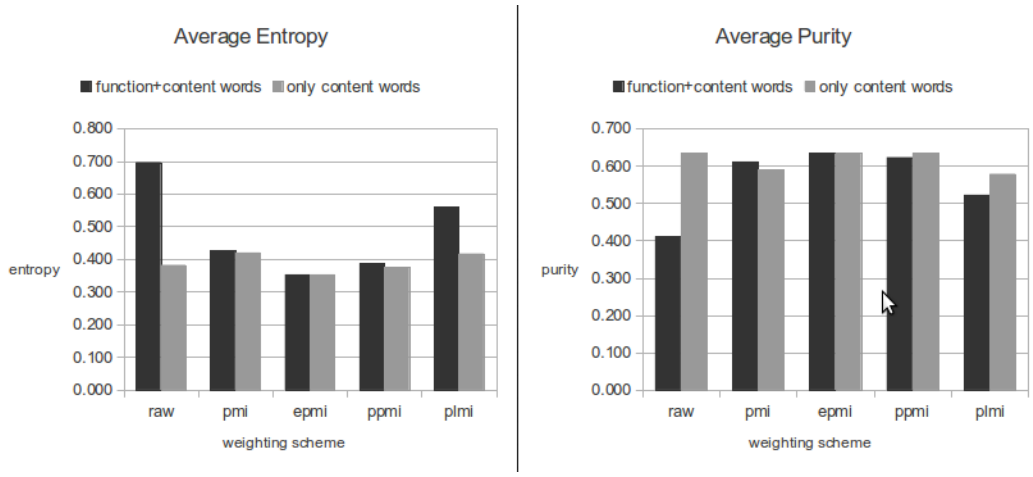


Figure 5.9: Verbs clustering comparison between CCG-DSM with both function and content words, and CCG-DSM with only content words

CCG-DSM with LMI-based weighting scheme. The clustering qualities of both mentioned models however are still not better than CCG-DSM with PMI-based weighting. Meanwhile, the CCG-DSM with PMI-based weighting seems not affected by the removal of function words.

Qualitative Evaluation

Cluster	Classes									Entropy	Purity
	exch[1]	moti[2]	chan[3]	body[4]	chan[5]	ment[6]	moti[7]	body[8]	comm[9]		
0	0	0	0	2	0	0	0	0	0	0.000	1.000
1	0	0	0	2	0	0	0	0	0	0.000	1.000
2	0	0	0	1	0	0	0	1	0	0.315	0.500
3	0	0	0	0	0	0	2	0	0	0.000	1.000
4	5	0	0	0	0	0	0	0	0	0.000	1.000
5	0	0	0	0	0	3	0	4	3	0.496	0.400
6	0	0	4	0	0	0	0	0	0	0.000	1.000
7	0	1	0	0	2	2	0	0	2	0.615	0.286
8	0	4	1	0	3	0	3	0	0	0.589	0.364

[1] exchange [2] motionDirection [3] changeState [4] bodyAction [5] changeLocation [6] mentalState
[7] motionManner [8] bodySense [9] communication

Table 5.13: Confusion matrix for 9-way verbs clustering

Table 5.13 exhibits the confusion matrix for 9-way verbs clustering, while Table 5.14 shows the descriptive features of each cluster. This clustering result is based on the result of CCG-based model using EPMI weighting scheme. The clusters that are successfully built according to the gold standard are cluster of verbs from *exchange* class (cluster 4) and cluster of verbs from *changeState* class (cluster 6).

As for the others, the model either makes unnecessary clusters for verbs belong to the same class, or fails to distinguish verbs belong to the different class. For example, the model separates verbs of *bodyAction* class which is related to body system such as "eat" and "drink" (cluster 0), from verbs of *bodyAction* class which is used to express emotion such as "smile" and "cry" (cluster 1). From both tables we can also see that the clusters with worst

Cluster	Descriptive features
0	**eat**_S[b]\NP 9.0%, **drink**_(S[b]\NP)/NP 7.0%, **eat**_(S[b]\NP)/NP 3.8%, **eat**_(S[dcl]\NP)/NP 3.5%, **alcohol**_N 2.7%
1	**smile**_N 4.9%, **make**_(S[dcl]\NP)/S[dcl] 3.1%, **shake**_(S[dcl]\NP)/NP 3.1%, **tear**_N 2.7%, **make**_(S[b]\NP)/S[dcl] 2.4%
2	**breath**_N/PP 12.2%, **smell**_N 9.4%, **breath**_N 4.6%, **lung**_N 3.6%, **oxygen**_N 2.6%
3	**bike**_N 3.6%, **ride**_(S[b]\NP)/NP 3.1%, **bicycle**_N 2.9%, **motorcycle**_N 2.9%, **ride**_(S[ng]\NP)/NP
4	**lender**_N 0.7%, **stake**_N/PP 0.6%, **tribute**_N 0.5%, **auction**_N 0.5%, **borrower**_N 0.4%
5	**password**_N/PP 0.4%, **like**_PP/S[dcl] 0.3%, **listen**_S[b]\NP 0.3%, **talk**_S[ng]\NP 0.3%, **talk**_S[b]\NP 0.2%
6	**inter**_S[pss]\NP 4.3%, **damaged**_N/N 3.3%, **wound**_S[pss]\NP 2.0%, **damage**_S[pss]\NP 1.6%, **Cemetery**_N 1.3%
7	**that**_S[bem]/S[b] 0.6%, **evidence**_N/(S[to]\NP) 0.2%, **password**_N/PP 0.2%, **out**_PP/NP 0.2%, **out**_((S\NP)\(S\NP))/NP 0.2%
8	**sharply**_(S\NP)\(S\NP) 0.3%, **of**_(PP\PR)/NP 0.2%, **trigger**_N 0.2%, **behind**_(S\NP)\(S\NP) 0.2%, **apart**_S[adj]\NP 0.2%

Table 5.14: Descriptive features for 9-way verbs clustering

qualities are the ones which have function words as the descriptive features (cluster 7 and cluster 8).

Cluster	Classes					Entropy	Purity
	exch[1]	moti[2]	chan[3]	body[4]	cogn[5]		
0	0	0	0	4	0	0.000	1.000
1	0	0	4	0	0	0.000	1.000
2	0	0	0	6	6	0.431	0.500
3	5	3	0	0	4	0.670	0.417
4	0	12	1	0	0	0.168	0.923

[1] exchange [2] motion [3] changeState [4] body [5] cognition

Table 5.15: Confusion matrix for 5-way verbs clustering

Cluster	Descriptive features
0	**eat**_S[b]\NP 4.4%, **breath**_N/PP 4.4%, **smell**_N 3.6%, **drink**_(S[b]\NP)/NP 3.1%, **eat**_(S[b]\NP)/NP 1.8%
1	**inter**_S[pss]\NP 4.3%, **damaged**_N/N 3.3%, **wound**_S[pss]\NP 2.0%, **damage**_S[pss]\NP 1.6%, **Cemetery**_N 1.3%
2	**smile**_N 0.4%, **and**_(S\NP)/(S\NP) 0.3%, **talk**_S[b]\NP 0.3%, **talk**_S[ng]\NP 0.3%, **make**_(S[dcl]\NP)/S[dcl] 0.3%
3	**that**_S[bem]/S[b] 0.3%, **lender**_N 0.2%, **Broadcasting**_N 0.2%, **kindly**_(S\NP)/(S\NP) 0.1%, **stake**_N/PP 0.1%
4	**sharply**_(S\NP)\(S\NP) 0.2%, **of**_(PP\PR)/NP 0.2%, **bike**_N 0.2%, **behind**_(S\NP)\(S\NP) 0.2%, **trigger**_N 0.2%

Table 5.16: Descriptive features for 5-way verbs clustering

Similarly, the confusion matrix for 5-way verbs clustering as well as the descriptive features for each cluster are described by Table 5.15 and Table 5.16 respectively.

The model once again makes distinction between verbs of *body* class which is related to body system (cluster 0), and verbs of *body* class which is used to express emotion (cluster 2). However, the model is quite successful in grouping most of the verbs from *motion* class

in one cluster (cluster 4), and almost all of the verbs from *changeState* class in one cluster (cluster 1).

Chapter 6

Conclusion

Semantics representation of natural language has been one of the problem in computational linguistics that many researchers tried to tackle. Distributional Semantic Models (DSMs) exploit the idea that the meaning of words is closely connected to the statistics of word usage, and have become increasingly popular in cognitive science.

The main idea of this thesis is to propose a new approach to exploit syntactic information within distributional semantics, which is incorporating Combinatory Categorial Grammar (CCG) categories in the linguistic context used to represent the meaning of a word. We have given overview about the existing related works and their possible limitations, that became the motivations behind this work.

We introduced a general framework to build enhanced DSM with CCG categories, based on the standard DSMs framework. The framework includes selecting target words of interest, defining linguistic context which is enriched with CCG categories, constructing co-occurrence matrix using the CCG parsed corpus, and apply various weighting schemes.

Through various experiments of word categorization task, we could answer the research questions which have been the foundation of this thesis. The first one is regarding the performance of CCG-based DSM compared with other types of DSM.

In the word clustering task in general, it is shown that CCG-based DSM which employs supertags (*i.e.*, CCG categories) is generally better than structured DSM with traditional less informative part-of-speech (POS) tags, and most certainly outperform unstructured DSM that do not provide any syntactic information in the linguistic context.

In the task of verbs clustering, CCG-based DSM performs better than structured DSM which uses dependency relation to define linguistic context and considered to be the state-of-the-art model. This proves that syntactic categories, in this case CCG categories, play a great role in defining the context of verbs meaning. However, in concrete nouns clustering, the quality of clusters produced by CCG-based DSM is still not as good as dependency-based DSM.

Regarding the weighting schemes, we found that PMI-based weighting schemes, especially PPMI and EPMI, perform better than the plain raw co-occurrence frequency values and LMI-based weighting schemes. LMI-based weighting schemes have similar performance with raw frequency since raw frequency contributes significantly in the equation. The overall best performance for concrete nouns clustering is achieved by CCG-based model with PPMI weighting, while the best performance for verbs clustering is obtained using CCG-based model with EPMI weighting.

The above explained experiments are done using the CCG-based DSM with the sentence as context window, meaning that during the construction of co-occurrence matrix, two words are considered to co-occur if they are in the same sentence. To answer the second research question regarding the selection of window size, we build another CCG-based DSM that

uses smaller context window size to determine the co-occurrence counts.

We found that using smaller window size, the clusters' quality produced in nouns clustering task is significantly improved especially for models with raw frequency and PLMI weighting schemes, perhaps due to the absence of unrelated grammatical words exist in larger context window. The performance of models with EPMI and PPMI weighting seems not affected by the reduced window size, even slightly declines. However, the opposite phenomenon happened in verbs clustering. This shows that the meaning of verbs is more closely related to the adjacent words.

The last research question that we tried to answer is what will be the outcome of including function words in the linguistic context of CCG-based DSM. We found that the performance of CCG-based DSM which use raw co-occurrence frequency and LMI-based weighting decreases as we include function words in the co-occurrence matrix. Whereas for the PMI-based models, the inclusion gives almost no impact perhaps due to their nature to give little importance to high frequency words, which is usually the characteristic of function words.

Finally, with some improvements in the way of using CCG categories to enrich the linguistic context, CCG-based DSM shows a great promise to be able to give better distributional meaning to words. There are several topics regarding the use of CCG in distributional semantics approach that could be explored further for future research.

Distributional Meaning of Function Words

As previously mentioned, CCG-based DSM shows a better result in verb categorization task compared with dependency-based DSM, giving the evidence to the importance of including syntactic categories, in this case CCG categories, within the context vector to define the meaning of verbs. Furthermore, the performance of CCG-based DSM improves when using smaller context window size. This phenomenon might be related to the nature of verbs in CCG as the type of functions that takes arguments, either the word before or after, to result in another CCG categories.

Given that grammatical words such as quantifiers and determiners have the similar type of CCG categories as verbs, we could assume that the same aspects will apply. Hence, this approach might be used to define the semantics of function or grammatical words, which is a difficult task due to the sparseness of function words that tend to co-occur with a lot of words in high frequency within the corpus.

CCG for Compositionality in DSM

The main focus in this thesis is how to represent the distributional meaning of words in isolation. Recently, researches in distributional semantic approach started to move toward the direction of representing the composition of words, *i.e.*, phrases or sentences as the higher aim.

CCG categories and combinatory rules that are used to combine words together might be useful to define linguistic context of words composition. Recent work of Baroni and Zamparelli (2010) shows that defining the meaning of adjective-noun composition can be approached with distributional semantic models by representing nouns as vectors and adjectives as functions (encoded as matrices) over nominal vectors.

If one atomic CCG category, take NP as an example, can be viewed as a vector (one-order tensor), then if we have an intransitive verb with category S\NP we will have a matrix (two-order tensor) because it takes atomic category NP to be another atomic category S. Applying the same arguments then we can have categories which are three-order tensor and so on. Using this nature of CCG categories, one can explore the representation of more complex compositions than previously mentioned adjective-noun composition.

Appendix A

Appendix

A.1 Function Words

The total number of function words found is 202 words, which have POS tags listed in Table A.1, with the addition of primary verbs *be*, *do*, and *have*. The complete list can be found in Table A.2.

CC	coordinating conjunction	PRP$	possessive pronoun
CD	cardinal number	RP	particle
DT	determiner	SYM	symbol
EX	existential	TO	*to*
IN	preposition	UH	interjection
LS	list item marker	WDT	wh-determiner
MD	modal	WP	wh-pronoun
PDT	predeterminer	WP$	possessive wh-pronoun
POS	possessive ending	WRB	wh-adverb
PRP	personal pronoun		

Table A.1: POS tags of function words

CC	and	DT	a	IN	aboard	into	PRP$	her
	both		all		about	like		his
	but		an		above	near		its
	either		another		across	nearest		my
	neither		any		after	next		our
	nor		anybody		against	of		their
	or		anyone		ago	off		your
	plus		anything		albeit	on	PRP	I
	yet		each		along	once		he
CD	billion		every		alongside	onto		herself
	eight		everybody		although	out		himself
	eighteen		everyone		amid	outside		it
	eleven		everything		among	over		itself
	fifteen		half		amongst	past		myself
	five		no		around	per		ourselves
	four		nobody		as	since		s
	fourteen		nothing		at	so		she
	hundred		some		atop	than		themselves
	million		somebody		because	though		they
	nine		someone		before	through		we
	one		something		behind	throughout		you
	seven		that		below	till		yourself
	six		the		beneath	toward	WDT	whatever
	ten		these		beside	towards		which
	thirteen		this		besides	under		whichever
	thirty		those		between	underneath	WP$	whose
	thousand	EX	there		beyond	unless	WP	what
	three	PDT	such		by	unlike		who
	twelve	TO	to		despite	until		whoever
	twenty	MD	can		down	up		whom
	two		could		download	upon	WRB	how
	zero		may		during	via		when
RP	aside		might		en	whereas		whenever
	away		must		except	whether		where
	back		need		for	while		whereby
	forward		ought		from	whilst		wherever
UH	hello		shall		if	with		why
	oh		should		in	within		
	please		will		inside	without		
	well		would		instead	worth		
	yeah	VB	be					
	yes		do					
			have					

Table A.2: List of function words according to their POS tags

A.2 Co-occurrence Counts Extraction Implementation

The algorithm to extract co-occurrence counts is implemented in C++ language, as follows:

```
#include <stdio.h>
#include <stdlib.h>
#include <regex.h>
#include <string.h>
#include <map>
#include <tr1/unordered_map>
#include <vector>
#include <utility>
#include <iostream>
#include <fstream>
#include <cstdio>

using namespace std;
using namespace std::tr1;

typedef short int sint;
typedef unsigned short int usint;

sint word_in_target(string lemma, string pos, vector<string> target_words, unordered_map<string, pair<usint, usint> > index) {
    usint start=0, end=0;
    string letter = lemma.substr(0,1) + "." + pos;
    start=index[letter].first;
    end=index[letter].second;
    if (end != 0) {
        for (int i=start; i<end; i++) {
            if (strcmp(lemma.c_str(), target_words[i].substr(0,target_words[i].find("_")).c_str()) == 0 &&
                strcmp(pos.c_str(), target_words[i].substr((target_words[i].find("_"))+1).c_str()) == 0) {
                return i;
            }
        }
    }
    return -99;
}

sint word_in_dimension(string word, vector<string> dimension, unordered_map<char, pair<usint, usint> > index) {
    usint start = index[word[0]].first;
    usint end = index[word[0]].second;
    for (int i=start; i<end;i++) {
        if (strcmp(dimension[i].c_str(), word.c_str()) == 0) return i;
    }
    return -99;
}

bool valid_word(string word) {
    int dash=0;
    for (int i=0; i< word.size(); i++) {
        if (!isalpha(word[i]) && word[i] != '-') return false;
        if (word[i] == '-') dash++;
    }
    if (dash > 1 || word[0] == '-' || word[word.size()-1] == '-') return false;
    return true;
}

int main(int argc, const char *argv[]) {
    string line;
    const char *pattern1 = "lemma=\"(.+?)\" pos=\"(.+?)\" chunk=\".+?\" entity=\".+?\" cat=\"(.+?)\" />";
    regex_t p1;
    regmatch_t match[4];
    int status;
    string lemma, pos, ccg, word, poss, letter, tag;
    bool push;
    unordered_map<string, pair<usint, usint> > tindex;
    unordered_map<char, pair<usint, usint> > dindex;
    vector<string> target_words;
    vector<string> dimensions;
    vector<pair<sint, sint> > sentence;
    vector< vector<usint> > counts;
    string fname, foutname;
    sint idxstart, idxend, tidx, didx, sidx;
    usint window_size = 0;

    if (argc > 6) {
```

```
// open the ccg parsed file
fname = argv[1];
ifstream file(fname.c_str());

// which tags will be used: CCG categories ("ccg") or POS tags ("pos")
tag = argv[2];

// open the target words file
ifstream ftarget(argv[3]);
while (getline(ftarget,line))
    target_words.push_back(line);
ftarget.close();

// open the target words index file
ifstream ftindex(argv[4]);
while (getline(ftindex,line)) {
    letter = line.substr(0, line.find("\t"));
    idxstart = atoi(line.substr(line.find("\t")+1, line.find("-")).c_str());
    idxend = atoi(line.substr(line.find("-")+1).c_str());
    tindex.insert(make_pair(letter, make_pair(idxstart, idxend)));
}
ftindex.close();

// open the dimensions file
ifstream fdimension(argv[5]);
while (getline(fdimension,line))
    dimensions.push_back(line);
fdimension.close();

// open the dimensions index file
ifstream fdindex(argv[6]);
while (getline(fdindex,line)) {
    idxstart = atoi(line.substr(line.find("\t")+1, line.find("-")).c_str());
    idxend = atoi(line.substr(line.find("-")+1).c_str());
    dindex.insert(make_pair(line[0], make_pair(idxstart, idxend)));
}
fdindex.close();

// window size is specified
if (argc > 7) {
    window_size = atoi(argv[7]);
}

foutname = tag;
foutname += "-";
foutname += argv[1];
ofstream fout (foutname.c_str());
cout << foutname << endl;

if (file != NULL) {
    counts.resize(target_words.size(), vector<usint>(dimensions.size(), 0));

    if (regcomp(&p1, pattern1, REG_ICASE | REG_EXTENDED)) {
        printf("Pattern 1 did not compile.\n");
        return(1);
    }

    while (getline(file,line)) {

        if (strcmp(line.c_str(), "</ccg>") != 0) {
            status = regexec(&p1, line.c_str(), 4, match, 0);
            if (!status) {
                lemma = line.substr(match[1].rm_so, match[1].rm_eo-match[1].rm_so);
                pos = line.substr(match[2].rm_so, match[2].rm_eo-match[2].rm_so);
                ccg = line.substr(match[3].rm_so, match[3].rm_eo-match[3].rm_so);

                if (window_size == 0) { // window size is not specified, sentence as context window
                    push = false;
                } else {
                    push = true;
                }

                if (valid_word(lemma)) {
                    if (strcmp(tag.c_str(), "ccg") == 0)  // CCG categories in dimensions
                        word = lemma + "_" + ccg;
                    else  // POS tag in dimensions
                        word = lemma + "_" + pos;
```

```
                        didx = word_in_dimension(word, dimensions, dindex);
                        if (window_size == 0) { // window size is not specified, sentence as context window
                            if (didx != -99) push = true;
                        }

                        if (pos[0]=='N' && pos[1]=='N') poss = "n";
                        else if (pos[0]=='V') poss = "v";
                        else if (pos[0]=='J') poss = "jj";
                        else poss = "";
                        if (strcmp(poss.c_str(), "") != 0) {
                            tidx = word_in_target(lemma, poss, target_words, tindex);
                            if (window_size == 0) { // window size is not specified, sentence as context window
                                if (tidx != -99) push = true;
                            }
                        } else { tidx = -99; }

                        if (push) sentence.push_back(pair<sint, sint>(tidx, didx));
                    }
                }

            } else { //end of sentence
                for (int i=0; i<sentence.size(); i++) {
                    tidx = sentence[i].first;
                    if (tidx != -99) { // target word
                        if (window_size == 0) { // window size is not specified, sentence as context window
                            for (int j=0; j<sentence.size(); j++) {
                                didx = sentence[j].second;
                                if (j!= i && didx != -99) { // dimensions
                                    ++counts[tidx][didx];
                                }
                            }
                        } else { // window size is specified
                            sidx = i;
                            for (int w=0; w<window_size; w++) {
                                sidx--;
                                if (sidx >= 0) {
                                    didx = sentence[sidx].second;
                                    if (didx != -99) ++counts[tidx][didx];
                                }
                            }
                            sidx = i;
                            for (int w=0; w<window_size; w++) {
                                sidx++;
                                if (sidx < sentence.size()) {
                                    didx = sentence[sidx].second;
                                    if (didx != -99) ++counts[tidx][didx];
                                }
                            }
                        }
                    }
                }
                sentence.clear();

            }
        }
        cout << "write to file..." << endl;
        for (sint row = 0; row < target_words.size(); row++) {
            for (sint col = 0; col < dimensions.size(); col++) {
                if (counts[row][col] > 0) fout << row << '\t' << col << '\t' << counts[row][col] << '\n';
            }
        }

        file.close();
        fout.close();
        regfree (&p1);
    }
}

    return 0;
}
```

The co-occurrence counts for each chunk file is saved into temporary file. Another algorithm is then run to combine all of them into one big co-occurrence matrix, which is implemented as follows:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <vector>
#include <utility>
#include <iostream>
#include <fstream>
#include <dirent.h>
#include <cstdio>
#include <ctype.h>

using namespace std;

#define TARGET_WORDS_SIZE 20001
#define DIMENSIONS_SIZE 20000

typedef short int sint;
typedef unsigned short int usint;

int main(int argc, const char *argv[]) {
    string line;
    string fname, foutname;
    string filepath, filename;
    vector<string> target_words;
    vector<string> dimensions;
    vector< vector<usint> > counts;
    sint idx1, idx2, tidx, didx;
    usint count;

    if (argc > 3 && strcmp(argv[2], "") != 0 && strcmp(argv[3], "") != 0) {
        //open the target words file
        ifstream ftarget(argv[2]);
        while(getline(ftarget,line))
            target_words.push_back(line);
        ftarget.close();

        //open the dimensions file
        ifstream fdimension(argv[3]);
        while(getline(fdimension,line))
            dimensions.push_back(line);
        fdimension.close();
    }

    if (argc > 1) {
        //open the directory of co-occurrence counts files
        DIR *pdir = NULL;
        pdir = opendir (argv[1]);

        //allocate the matrix
        counts.resize(TARGET_WORDS_SIZE, vector<usint>(DIMENSIONS_SIZE, 0));

        if (pdir == NULL) { //pdir is not directory, open the file
            ifstream file(argv[1]);
            if (file != NULL) {
                while (getline(file,line)) {
                    //update the co-occurrence count value
                    idx1 = line.find("\t");
                    idx2 = line.find_last_of("\t");
                    tidx = atoi(line.substr(0,idx1).c_str());
                    didx = atoi(line.substr(idx1+1, idx2-idx1-1).c_str());
                    count = atoi(line.substr(idx2+1).c_str());
                    counts[tidx][didx] += count;
                }
                file.close();
            }
            foutname = argv[1];
        } else {
            struct dirent *pent = NULL;
            while (pent = readdir (pdir))
            {
                if (pent == NULL) { //if pent has not been initialised correctly
                    cout << "ERROR! pent could not be initialised correctly" << endl;
                    return 0;
                }

                if (strcmp(pent->d_name,".") != 0 && strcmp(pent->d_name,"..") != 0) {
                    //open the co-occurrence counts file
```

```
                fname = argv[1];
                fname += "/";
                fname += pent->d_name;
                cout << fname << endl;

                ifstream file(fname.c_str());
                if (file != NULL) {
                    while (getline(file,line)) {
                        //update the co-occurrence count value
                        idx1 = line.find("\t");
                        idx2 = line.find_last_of("\t");
                        tidx = atoi(line.substr(0,idx1).c_str());
                        didx = atoi(line.substr(idx1+1, idx2-idx1-1).c_str());
                        count = atoi(line.substr(idx2+1).c_str());
                        counts[tidx][didx] += count;
                    }
                    file.close();
                }
            }
        }
        foutname = "combined-";
        filepath = argv[1];
        filename = filepath.substr(filepath.find_last_of("/")+1);
        foutname += filename;
    }

    if (target_words.size() > 0 && dimensions.size() > 0) foutname += ".mat";
    ofstream fout (foutname.c_str());

    cout << "write to file... " << foutname << endl;
    for (sint row = 0; row < TARGET_WORDS_SIZE; row++) {
        for (sint col = 0; col < DIMENSIONS_SIZE; col++) {
            if (target_words.size() > 0 && dimensions.size() > 0) {
                if (counts[row][col] > 0)
                    fout << target_words[row] << '\t' << dimensions[col] << '\t' << counts[row][col] << '\n';
            } else {
                if (counts[row][col] > 0)
                    fout << row << '\t' << col << '\t' << counts[row][col] << '\n';
            }
        }
    }

    fout.close();
    closedir (pdir);
}

return 0;
}
```

References

Almuhareb, A., & Poesio, M. (2005). Concept learning and categorization from the web. In *In proceedings of cogsci* (pp. 103–108). Stresa.

Bangalore, S., & Joshi, A. K. (1999, June). Supertagging: an approach to almost parsing. *Comput. Linguist.*, *25*(2), 237–265. Available from `http://dl.acm.org/citation.cfm?id=973306.973310`

Bar-Hillel, Y. (1953). A quasi-arithmetical notation for syntactic description. *Language*, *29*, 47-58.

Baroni, M., & Lenci, A. (2010, October 15). Distributional memory: A general framework for corpus-based semantics. *Computational Linguistics*, *36*(4), 673–721. Available from `http://dx.doi.org/10.1162/coli_a_00016`

Baroni, M., & Zamparelli, R. (2010). Nouns are vectors, adjectives are matrices: representing adjective-noun constructions in semantic space. In *Proceedings of the 2010 conference on empirical methods in natural language processing* (pp. 1183–1193). Stroudsburg, PA, USA: Association for Computational Linguistics. Available from `http://dl.acm.org/citation.cfm?id=1870658.1870773`

Bos, J., Clark, S., Steedman, M., Curran, J. R., & Hockenmaier, J. (2004). Wide-coverage semantic representations from a ccg parser. In *Proceedings of the 20th international conference on computational linguistics*. Stroudsburg, PA, USA: Association for Computational Linguistics. Available from `http://dx.doi.org/10.3115/1220355.1220535`

Buitelaar, P., & Magnini, B. (2005). Ontology learning from text: An overview. In *In paul buitelaar, p., cimiano, p., magnini b. (eds.), ontology learning from text: Methods, applications and evaluation* (pp. 3–12). IOS Press.

Bullinaria, J., & Levy, J. (2007). Extracting semantic representations from word co-occurrence statistics: A computational study. *Behavior Research Methods*(3), 510.

Chomsky, N. (1957). *Syntactic structures*. The Hague: Mouton.

Church, K. W., & Hanks, P. (1989). Word association norms, mutual information, and lexicography. In *Proceedings of the 27th annual meeting on association for computational linguistics* (pp. 76–83). Stroudsburg, PA, USA: Association for Computational Linguistics. Available from `http://dx.doi.org/10.3115/981623.981633`

Clark, S., & Curran, J. R. (2003). Log-linear models for wide-coverage ccg parsing. In *Proceedings of the 2003 conference on empirical methods in natural language processing* (pp. 97–104). Stroudsburg, PA, USA: Association for Computational Linguistics. Available from `http://dx.doi.org/10.3115/1119355.1119368`

Clark, S., & Curran, J. R. (2004). The importance of supertagging for wide-coverage ccg parsing. In *Proceedings of the 20th international conference on computational linguistics*. Stroudsburg, PA, USA: Association for Computational Linguistics. Available from `http://dx.doi.org/10.3115/1220355.1220396`

Crouch, C. J. (1988). A cluster-based approach to thesaurus construction. In *Proceedings of the 11th annual international acm sigir conference on research and development in information retrieval* (pp. 309–320). New York, NY, USA: ACM. Available from `http://doi.acm.org/10.1145/62437.62467`

Curran, J. R., & Moens, M. (2002). Improvements in automatic thesaurus extraction. In *Proceedings of the acl-02 workshop on unsupervised lexical acquisition - volume 9* (pp. 59–66). Stroudsburg, PA, USA: Association for Computational Linguistics. Available from http://dx.doi.org/10.3115/1118627.1118635

Dalrymple, M. (2001). *Lexical functional grammar.* Academic Press. Available from http://books.google.co.uk/books?id=chEmQAAACAAJ

Debusmann, R. (2001). *A declarative grammar formalism for dependency grammar.* Unpublished master's thesis, Saarland University.

Denhière, G., & Lemaire, B. (2004). *A computational model of children's semantic memory.* Lawrence Erlbaum Associates. Available from http://cogprints.org/3777/

Dunning, T. (1993, March). Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, *19*(1), 61–74.

Evert, S. (2004). The statistics of word cooccurrences: word pairs and collocations. *Unpublished doctoral dissertation, Institut f*
"ur maschinelle Sprachverarbeitung, Universit
"at Stuttgart.

Ferraresi, A., Zanchetta, E., Baroni, M., & Bernardini, S. (2008). Introducing and evaluating ukwac, a very large web-derived corpus of english. In *In proceedings of the 4th web as corpus workshop (wac-4.*

Grefenstette, G. (1994). *Explorations in automatic thesaurus discovery.* Norwell, MA, USA: Kluwer Academic Publishers.

Griffiths, T. L., Tenenbaum, J. B., & Steyvers, M. (2007). Topics in semantic representation. *Psychological Review*, *114*, 2007.

Harris, Z. (1954). Distributional structure. *Word*, *10*(23), 146–162.

Hockenmaier, J. (2003). *Data and Models for Statistical Parsing with Combinatory Categorial Grammar.* Unpublished doctoral dissertation, University of Edinburgh.

Honnibal, M., Curran, J. R., & Bos, J. (2010). Rebanking ccgbank for improved np interpretation. In *Proceedings of the 48th annual meeting of the association for computational linguistics* (pp. 207–215). Stroudsburg, PA, USA: Association for Computational Linguistics. Available from http://dl.acm.org/citation.cfm?id=1858681.1858703

Jones, K. S. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, *28*, 11–21.

Joshi, A. K. (1985). Tree adjoining grammars: how much context-sensitivity is required to provide reasonable structural descriptions? In D. R. Dowty, L. Karttunen, & A. Zwicky (Eds.), *Natural language parsing* (pp. 206–250). Cambridge: Cambridge University Press.

Kanejiya, D., Kumar, A., & Prasad, S. (2003). Automatic evaluation of students' answers using syntactically enhanced lsa. In *Proceedings of the hlt-naacl 03 workshop on building educational applications using natural language processing - volume 2* (pp. 53–60). Stroudsburg, PA, USA: Association for Computational Linguistics. Available from http://dx.doi.org/10.3115/1118894.1118902

Karypis, G. (2003). CLUTO: A Clustering Toolkit [Computer software manual]. Available from http://glaros.dtc.umn.edu/gkhome/cluto/cluto/download

Laham, D. (2000). *Automated Content Assessment of Text Using Latent Semantic Analysis to Simulate Human Cognition.* Unpublished doctoral dissertation, University of Colorado.

Landauer, T. K., & Dumais, S. T. (1997). Solution to Plato's Problem: The Latent Semantic Analysis Theory of Acquisition, Induction and Representation of Knowledge. *Psychological Review*(104).

Lenci, A. (2008). Distributional semantics in linguistic and cognitive research. *From context to meaning: distributional models of the lexicon in linguistics and cognitive science, Italian Journal of Linguistics*, *20*(1), 1–31.

Lenci, A. (2010). The life cycle of knowledge. *Ontologies and the Lexicon. A Natural Language Processing Perspective*, 241–257.

Lin, D. (1998). Automatic retrieval and clustering of similar words. In *Proceedings of the 36th annual meeting of the association for computational linguistics and 17th international conference on computational linguistics - volume 2* (pp. 768–774). Stroudsburg, PA, USA: Association for Computational Linguistics. Available from `http://dx.doi.org/10.3115/980691.980696`

Lin, D., & Pantel, P. (2001). Dirt - discovery of inference rules from text. In *In proceedings of the acm sigkdd conference on knowledge discovery and data mining* (pp. 323–328).

Lund, K., & Burgess, C. (1996). Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instruments & Computers*.

Melčuk, I. (1988). *Dependency syntax: Theory and practice*. State University Press of New York. Available from `http://books.google.it/books?id=diq29vrjAa4C`

Meyers, A., Reeves, R., Macleod, C., Szekely, R., Zielinska, V., Young, B., et al. (2004, May). The NomBank Project: An Interim Report. In A. Meyers (Ed.), *Hlt-naacl 2004 workshop: Frontiers in corpus annotation* (pp. 24–31). Boston, Massachusetts, USA: Association for Computational Linguistics.

Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., & Miller, K. (1990). Wordnet: An on-line lexical database. *International Journal of Lexicography*, *3*, 235–244.

Miller, G. A., & Charles, W. G. (1991). Contextual correlates of semantic similarity. *Language and Cognitive Processes*, *6*(1), 1–28. Available from `http://dx.doi.org/10.1080/01690969108406936`

Mitchell, J., & Lapata, M. (2010). Composition in Distributional Models of Semantics. *Cognitive Science*, *34*(8), 1388–1429.

Niwa, Y., & Nitta, Y. (1994). Co-occurrence vectors from corpora vs. distance vectors from dictionaries. In *Proceedings of the 15th conference on computational linguistics - volume 1* (pp. 304–309). Stroudsburg, PA, USA: Association for Computational Linguistics. Available from `http://dx.doi.org/10.3115/991886.991938`

Padó, S., & Lapata, M. (2007, June). Dependency-based construction of semantic space models. *Computational Linguistics*, *33*(2), 161–199. Available from `http://dx.doi.org/10.1162/coli.2007.33.2.161`

Palmer, M., Gildea, D., & Kingsbury, P. (2005, March). The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, *31*(1), 71–106. Available from `http://dx.doi.org/10.1162/0891201053630264`

Pantel, P., & Lin, D. (2002). Discovering word senses from text. In *Proceedings of the eighth acm sigkdd international conference on knowledge discovery and data mining* (pp. 613–619). New York, NY, USA: ACM. Available from `http://doi.acm.org/10.1145/775047.775138`

Proudian, D., & Pollard, C. (1985). Parsing head-driven phrase structure grammar. In W. C. Mann (Ed.), *Acl* (p. 167-171). ACL.

Rapp, R. (2003). Word sense discovery based on sense descriptor dissimilarity. In *Proceedings of the ninth machine translation summit.* Available from `http://www.mt-archive.info/MTS-2003-Rapp.pdf`

Rapp, R. (2004). A freely available automatically generated thesaurus of related words. In *Proceedings of language resources and evaluation (lrec) 2004* (pp. 395–398).

Rothenhäusler, K., & Schütze, H. (2009). Unsupervised classification with dependency based word spaces. In *Proceedings of the workshop on geometrical models of natural language semantics* (pp. 17–24). Stroudsburg, PA, USA: Association for Computational Linguistics. Available from `http://dl.acm.org/citation.cfm?id=1705415.1705418`

Sahlgren, M. (2006). *The Word-Space Model: using distributional analysis to represent syntagmatic and paradigmatic relations between words in high-dimensional vector spaces.* Unpublished doctoral dissertation, Stockholm University.

Schütze, H. (1998, March). Automatic word sense discrimination. *Comput. Linguist.*, *24*(1), 97–123. Available from http://dl.acm.org/citation.cfm?id=972719.972724

Shannon, C. E. (1948). A mathematical theory of communication. *Bell system technical journal*, *27*.

Steedman, M. (2000). *The syntactic process*. Cambridge, MA, USA: MIT Press.

Steedman, M., & Baldridge, J. (2011). Combinatory categorial grammar. In *Non-transformational syntax* (pp. 181–224). Wiley-Blackwell. Available from http://dx.doi.org/10.1002/9781444395037.ch5

Sugayama, K., & Hudson, R. (2006). *Word grammar: Perspectives on a theory of language structure*. Bloomsbury. Available from http://books.google.it/books?id=p-t-m-gFe9kC

Tesnière, L. (1959). *Eléments de syntaxe structurale*. Paris: Klincksieck.

Turney, P. D. (2006). Similarity of semantic relations. *Computational Linguistics*, *32*, 379–416.

Turney, P. D., & Littman, M. L. (2003, October). Measuring praise and criticism: Inference of semantic orientation from association. *ACM Trans. Inf. Syst.*, *21*(4), 315–346. Available from http://doi.acm.org/10.1145/944012.944013

Turney, P. D., & Pantel, P. (2010, March 4). From Frequency to Meaning: Vector Space Models of Semantics. *J. Artif. Intell. Res. (JAIR)*, *37*, 141–188. Available from http://dx.doi.org/10.1613/jair.2934

Widdows, D. (2003). Unsupervised methods for developing taxonomies by combining syntactic and statistical information. In *Proceedings of the 2003 conference of the north american chapter of the association for computational linguistics on human language technology - volume 1* (pp. 197–204). Stroudsburg, PA, USA: Association for Computational Linguistics. Available from http://dx.doi.org/10.3115/1073445.1073481

Zhao, Y., & Karypis, G. (2001). *Criterion functions for document clustering: Experiments and analysis.* Available from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.16.6872