Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Automatic Vocabulary Generation

by Latent Semantic Analysis and a Recommender System

Bachelor-Thesis

Simon Wehrli

siwehrli@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Tobias Langner, Philipp Brandes
Prof. Dr. Roger Wattenhofer

February 8, 2014

Acknowledgements

I would like to thank my supervisors Tobias Langner and Philipp Brandes, who gave me support during the whole project and helped me overcome obstacles. I would like to thank Michael Langner for his dedicated and motivating inputs. I am grateful to Professor Roger Wattenhofer for offering me such an interesting topic to investigate.

Abstract

In this thesis we developed a system that generates vocabulary lists for a desired semantic field. The user can select the field by just naming a few sample terms. The system uses Wikipedia as the text corpus and runs Probabilistic Latent Semantic Analysis on the corpus to represent the meanings of terms as vectors in a semantic space. A user's query is processed by searching for the nearest neighbors around the centre of his sample terms. Then, a clustering algorithm is run on the terms of the resulting list to assess their quality. Even though the presented semantic relatedness model is static, the user can provide feedback on generated vocabulary lists. The feedback is used by an integrated Recommender System to steadily improve existent lists and lists generated in the future.

Contents

Chapter 1

Introduction

Many people face the task of learning languages. In today's world it is of great advantage to be able to speak different languages. Learning vocabulary units builds the basis for both beginners and advanced learners. But the vocabulary they learn is totally different. While a beginner wants to learn the most important words such that he can start off communication, an advanced learner is interested in words of a particular field that he wants to delve into, for example because he needs it for his professional life.

There is a lot of research going on to explore different techniques of learning and teaching vocabulary. Researchers agree that learning a word is a cumulative process encompassing several encounters of the word [1]. Words should be used and repeated over different channels to embed them in the memory. A good learning progress is achieved if vocabulary is studied shortly before it is actively used. This requires that the vocabulary units are well coordinated with the practice.

Creating vocabulary units or exercises to train a particular vocabulary is a time-consuming work. Recently, computers became a helpful tool for teachers and learners. Online dictionaries and thesauri lead to faster results than traditional dictionaries. But they are focused on single terms. This project presents a tool to automatically generate whole vocabulary lists, that cover precisely the field the learner is interested in. There is no longer the need for manual selection of words. To decide which words a user might be interested in, our system automatically relates them. This is done by analyzing text sources created by humans and extracting relatedness information. This information is then used to find terms of a field a user describes by simple means, i.e. by some sample terms of the desired field. If the user reviews the list and disagrees with some terms, he can provide feedback to the system. His feedback is used to improve the system for further searches.

The generated vocabulary list cannot only be used by learners. It can help to understand or write text about a subject from a unfamiliar field. Teachers and language text book creators can use the generated vocabulary lists to build exercises. Another system may even use it to generate exercises automatically.

1.1 Background and Related Work

There are multiple ways how a term might be related to another term, e.g. as synonym, antonym or hypernym. These relations were collected and manually fixed in linguistic resources such as thesauri, ontologies and synonym dictionaries, and are successfully used in various applications, such as query expansion [2] and document categorization [3]. The problem is that for any relation type, a suitable resource with the right domain and language is required, but the construction of such resources is a time-consuming work.

Recently Wikipedia has become popular as a text corpus for automatic extraction of semantic relatedness information. There are various approaches to extract relations. The open-source system *Serelex*[1] [4] uses the abstracts of Wikipedia articles to build definition vectors of the covered concepts. A k-nearest neighbor algorithm and two semantic similarity measures (cosine similarity and Gloss Overlap) are then used to extract relations from the definition vectors. Another system, *BabelNet*, extends the same idea to a large multilingual semantic network [5]. It uses both encyclopedic text corpora and lexicographic knowledge from *WordNet*, a machine-readable lexical database in English [6]. The extension to other languages is supported by machine translation of the lexical database. But this is not applicable to all languages without additional lexical knowledge because machine translation is a complex problem itself.

Probabilistic Latent Semantic Analysis [7] is a technique for the analysis of co-occurrence data, which has applications in other areas apart from natural language processing. In this project it is applied to represent the meaning of terms as a weighted vector of latent concepts. Assessing the relatedness of terms amounts to comparing the corresponding vectors using conventional similarity measures. Explicit Semantic Analysis [8] is a variant that uses explicit concepts instead of latent concepts. The explicit concepts are derived from Wikipedia categories.

As mentioned, in any relation extraction process, one or several similarity measures are involved. One work compares five different similarity measures in different contexts [9], among them is also the cosine similarity used in this project. Another work explores hybrid semantic similarity measures [10].

Applications which use the extracted semantic relations include many fields, from improving Automatic Speech Recognition with WordNet-based semantic relatedness [11] to generating vocabulary questions [12]. This last project is one of the few that also aims on language learners as this project does.

[1] serelex.cental.be/

Chapter 2

Semantic Relation Extraction

The overall goal of semantic relation extraction is to annotate terms with information according to some model of *semantic relatedness*. The model must include a definition of what exactly we refer to as semantically related. In linguistics, it means that terms or symbols have similar meanings. Our purpose is the generation of vocabulary lists for arbitrary semantic fields, where we regard *semantic fields* as groups of terms that are often used together in natural languages. More precisely, we focus on written language. The simple reason is that it is much more difficult to get access to and to process spoken language. In many languages there is only a small difference between spoken and written language with regard to the used terms. This holds in particular for the English language, for which the system's prototype is built.[1]

In this chapter, we describe how we get from a text corpus to a interactive search for related terms. We start by defining a measure for semantic relatedness. We describe the chosen text corpus and how it is preprocessed. Then, we explain the theoretical basis of the relation extraction and how it is concretely applied. The extracted relatedness information is stored and used to search for related terms by a searching algorithm. Finally, we show how clustering is used to estimate the quality of the search result. We introduce the required algorithms and tools when they are first used.

2.1 Measures for Semantic Relatedness

The model of semantic relatedness must define the representation of the semantic relations. We adopt the widely used [8, 11, 10] representation with a number on a one-dimensional, ordinal scale since we are not distinguishing between different subtypes of relations and are measuring only the strength of the relatedness. We choose smaller values to signify stronger related terms.

[1] As a counterexample consider languages that do not even know any visual representation, like the original hawaiian language.

Definition 2.1. A *dissimilarity measure* is a function $\text{dis} : \mathcal{V} \times \mathcal{V} \to (0, +\infty]$, that given any pair out of a set of entities $\mathcal{V}$, returns an ordinal number that is smaller for more similar and larger for more dissimilar entities.

There is no dissimilarity measure which is directly applicable to terms. Hence we will need a meta description of terms:

Definition 2.2. A *semantic vector* is a vector $v \in \mathbb{R}^s$ associated with an item. The only requirement is that vectors of similar items are close with respect to a well defined measure.

Assume for now that for each term we have such a semantic vector (in Section 2.3 we will see how to find them). This allows us to use one of the dissimilarity measures defined over vectors in $\mathbb{R}^s$. Among them is also the familiar class of *distance metrics*.

Definition 2.3. A *distance metric* for a set of vectors $\mathcal{V}$ is a dissimilarity measure that satisfies the following three conditions for all $u, v, w \in \mathcal{V}$:

$$\begin{aligned} &\text{dis}(u, v) = 0 \Leftrightarrow u = v && \text{(identity)} \\ &\text{dis}(u, v) = \text{dis}(v, u), && \text{(symmetry)} \\ &\text{dis}(u, w) \leq \text{dis}(u, v) + \text{dis}(v, w) && \text{(triangle inequality)} \end{aligned}$$

The *Euclidean distance* is a widely used metric, but it is not the best choice for our semantic vectors, because in our model similar terms get mapped roughly to the same direction (not to the same point) in the semantic space. Hence we need a similarity measure based on the angle between vectors. This can lead to different results compared to the euclidean distance as sketched in Figure 2.1.

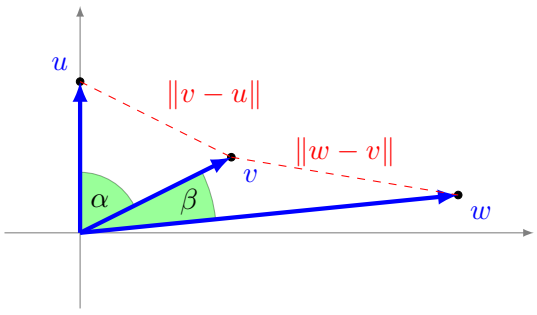

Figure 2.1: The Euclidean distance and the angle between vectors u, v, w compared. Even though the euclidean distances are related with $\|w - v\| > \|v - u\|$, the angles are in contrast related with $\beta < \alpha$.

A frequently used measure [13] that only consider angles is the *cosine similarity*. It is the best choice for similarity calculation between semantic vectors [9].

Definition 2.4. The *cosine similarity* and *dissimilarity* for vectors $u, v \in \mathbb{R}^s$ are given by

$$\text{cosSim}(u, v) = \frac{u \cdot v}{\|u\| \, \|v\|} = \cos(\alpha), \qquad \text{cosDis}(u, v) = 1 - \text{cosSim}(u, v). \tag{2.1}$$

where α is the angle between u and v and $\|\cdot\|$ is the L_2-norm.

Note that if u and v have no negative values, α cannot be greater than $90°$ and hence cosSim and cosDis ranges only from zero to one. As cosSim does not satisfy the identity condition (consider identical or orthogonal vectors) and the triangle inequality, it is not a metric. cosDis is not a metric either, since it is only a linear function of cosSim.

2.2 The Text Corpus

In principle, any text corpus can be used as the basis for text mining. For our purpose, the generation of term lists for given semantic fields, we have indeed some specific requirements. First, we should be able to relate as many terms as possible but at least the terms a native speaker has in its active vocabulary. Second, the text corpus should feature a deep structuring of terms according to the semantic field they are belonging to.

Wikipedia is a free online encyclopedia, which anyone can edit. Each article normally covers one concept. This separation of concepts produces a deep structure which is very valuable to extract semantic relations.[2] Because a lot of specialists write about their familiar subjects, an impressive high quality of articles is achieved. High quality is important because it implies that an article mentions all terms (and only terms) that are relevant for the covered concept. Wikipedia exists for all main languages to different extents, where the English one is the largest (nearly four million articles). Since no language specific processing is done, the system can easily be adapted for other languages. All articles are freely downloadable as XML dumps.[3] This properties already motivated other projects to use Wikipedia [14, 8].

2.2.1 Preprocessing

We would like to extract only the valuable information from the Wikipedia XML dumps. First we remove all Wikipedia markup language. This removes pictures, external URLs, special formatted sidebars and non-alphanumeric text such as phonetics. We also remove spacing such as newlines and punctuation characters.

[2] Compare it with a novel: in a novel it would be difficult to assume any particular relation between words in one chapter and words in another chapter.

[3] see dumps.wikimedia.org

What remains are the titles, subtitles and the plain text as a clean list of terms. We still keep up the separation between articles, hence the result from this steps is a set of text documents[4] $\mathcal{D} = \{d_1, \ldots, d_m\}$. Each text document d_j is itself represented as a list of terms.

The documents in $\mathcal{D}$ are now again filtered. As proposed by other projects [8], we first filter out all words of three and less characters because most of this short words are filler words. Then, all words not appearing in an English vocabulary[5] are filtered out. This is very helpful because words from other languages and names are not present in the filter vocabulary. Finally terms that appear very rarely (less than 10 times) in the text corpus are filtered out, because we are not able to relate them properly [9]. Now the documents $\mathcal{D}$ are ready for the latent semantic analysis.

2.3 Probabilistic Latent Semantic Analysis

Probabilistic Latent Semantic Analysis (PLSA) [7] is a technique for the analysis of co-occurrence data, which has applications in other areas apart from natural language processing. In this project it is applied to our text corpus to represent the meaning of terms as a weighted vector of latent concepts. Assessing the relatedness of terms amounts to comparing the corresponding vectors using conventional similarity measures. We focus on text data as the corpus even though it is applicable to other types of corpora. Thomas Hoffman describes all details in his paper about PLSA [7].

Let $\mathcal{T} = \{t_1, \ldots, t_n\}$ be a vocabulary of terms in some language and documents $\mathcal{D} = \{d_1, \ldots, d_m\}$ over the vocabulary $\mathcal{T}$, that means only terms from $\mathcal{T}$ may appear in the documents.[6] Ignoring the order in which terms occur in a document, we can build a so call *term-document matrix*.

Definition 2.5. The *term-document matrix* is defined as

$$\mathbf{C} \in \mathbb{N}^{n \times m}, \qquad \mathbf{C} = (\text{count}(t_i, d_j)), \tag{2.2}$$

where $\text{count}(t, d) \in \mathbb{N}$ returns how often t appears in d. Because documents contain few terms relative to the size of the vocabulary $\mathcal{T}$, $\mathbf{C}$ is sparse.

[4] From now on we will call them documents instead of articles follow the notation of referenced papers.

[5] The vocabulary of about 40 000 words was produced by the intersection between two freely available vocabularies: *twl06* from norvig.com/ngrams/ and *SCOWL* from wordlist.sourceforge.net/

[6] Obviously it is easy to collect all occurring terms $\mathcal{T}$ if $\mathcal{D}$ is given.

We could interpret rows of $\mathbf{C}$ as a representation of terms in a vector space and determine similarity between them. However, the idea of LSA is to reduce these m-dimensional rows first by mapping them to a vector space of lower dimensionality s, called *latent semantic space*. The hope is that we retrieve representational vectors that are no longer sparse and even allow for comparison of terms that do not appear together in a document. The mapping is restricted to linear functions and is derived from a latent variable model.

2.3.1 The Aspect Model

We only give the intuition here and refer to Thomas Hofmann's paper [7] for further details. The *Aspect Model* is a latent variable model and states the following assumptions:

- There is a set of aspects $\mathcal{A} = \{a_1, \ldots, a_s\}$, also called *latent classes*. There is a *latent class variable* $a \in \mathcal{A}$ which is an unobserved variable, i.e. cannot be measured. Since a is a categorical variable, it can only take one of the abstract classes in $\mathcal{A}$.
- Each observation of a term t in a document d is associated with a latent variable $a \in \mathcal{A}$. Intuitively, a is the *only* reason why t appears in d. There is no direct relation between terms and documents.[7] Aspects are contained in a document by a certain degree and aspects produce terms with a certain probability. Figure 2.2 visualizes the intuition.

We can interpret the conditional probability vectors $(\Pr(t|a_j))_{j=1,\ldots,s}$ and $(\Pr(d|a_j))_{j=1,\ldots,s}$ as an alternative description of terms and documents, respectively. This model can be parameterized with

$$\Pr(d,t) = \sum_{a \in \mathcal{A}} \Pr(a) \Pr(t|a) \Pr(d|a). \tag{2.3}$$

The model parameters are $\Pr(a)$, $\Pr(t|a)$ and $\Pr(d|a)$ for each $a \in \mathcal{A}, t \in \mathcal{T}$ and $d \in \mathcal{D}$.[8] The model fitting is done by a maximum likelihood estimation. Usually the Expectation Maximization (EM) algorithm is used. Note that this estimation algorithm is non-deterministic. Further details can be found in Section 3.2 of Hofmann's paper [7]. What matters for our project is the input/output behavior:

[7] t and d are so called *locally independent*.
[8] The number of latent classes $|\mathcal{A}|$ could also be fitted if not predefined.

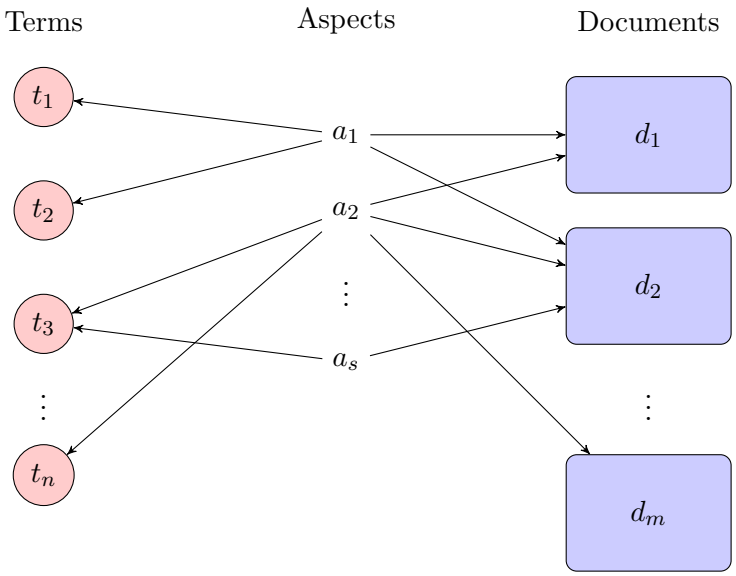

Figure 2.2: The Aspect Model states that aspects are contained in a document by a certain degree and aspects produce terms with a certain probability. Terms and documents have no direct relation.

Inputs are the term-document matrix $\mathbf{C}$ or a sparse representation thereof, the dimension s of the latent space, and the maximum number of steps that should be done to fit the model.

Outputs are the fitted model parameters. We will only need $\Pr(t|a)$ for each $a \in \mathcal{A}$, $t \in \mathcal{T}$.[9]

We can write the output probabilities $\Pr(t|a)$ as vectors $(\Pr(t|a_j))_{j=1,\dots,s}$ for some arbitrary but fixed order of the aspects a_j. If we recall the Definition 2.2 we see that this vectors can be interpreted as *semantic vectors* describing terms $t \in \mathcal{T}$. We store them for further use in a database. A lookup in the database is represented by the function

$$\begin{aligned} \omega : \mathcal{T} &\longrightarrow [0,1]^s, \\ \omega : t &\longmapsto \begin{bmatrix} \Pr(t|a_1) \\ \vdots \\ \Pr(t|a_s) \end{bmatrix}, \end{aligned}$$

which returns an s-dimensional semantic vector in the latent space for any $t \in \mathcal{T}$. The used implementation of PLSA returns all semantic vectors normalized with respect to the L_1-norm.

[9] We also get the other fitted model parameters, such as the vectors characterizing documents $(\Pr(d|a_j))_{j=1,\dots,s}$ and the global aspect weights $\Pr(a)$ for $a \in \mathcal{A}$.

2.3.2 Experiment

During the project development the semantic analysis was run with different parameters.[10] The parameters which subjectively produced the best results were then used to build the prototype described later. The intention of this section is to provide the necessary details to reproduce our results.

In the best run the following parameters where chosen: The number of read Wikipedia articles was limited to 20 000 (out of 4.5 millions) to avoid memory overflows. This restriction is notable since some semantic field are worse related even though Wikipedia would cover them. The dimension of the semantic vectors was set to 500, because longer vectors has shown to bring no better quality of the relations. The EM-algorithm was iterated 1000 times, after that the error did not decrease anymore. This settings led to a term-document matrix size of $(41\,305, 20\,000)$. The entire process took 75.5 hours on our machine.[11]

2.4 Semantic Nearest Neighbor Search

Given the terms $t_i \in \mathcal{T}$, the previous section explained how we extract their associated semantic vectors $\omega(t_i)$ in an s-dimensional semantic space. In this section we will discuss how an actual semantic search is done in this space.

To define the strength of semantic relatedness between any two terms t_1, t_2, we need to define a notion of *similarity* between them. As we motivated in Section 2.1, we do this over the associated semantic vectors $\omega(t_1)$ and $\omega(t_2)$. In Section 2.1, we argued why the *cosine similarity* is well suited for this purpose.

The input of a search query made by a user (or another system component) consists of k query terms q_j, $j = 1, \ldots, k$ and the desired count of related terms ℓ to be returned. Implicitly, ℓ defines the search radius. It is assumed that the query terms are present in the database, hence we have semantic vectors $u_i = \omega(q_j)$ available. We would like to find the nearest neighbors of these vectors. There are two possible approaches to handle multiple query terms:

1. Use each query term vector u_j as a separate centre point to start a nearest neighbor search off,

2. combine the query terms to a single centre by an averaging function and do a single nearest neighbor search starting at this centre point.

[10] For this project a python implementation of PLSA from www.mblondel.org/journal/ was used.

[11] Machine: Quad-Core AMD Opteron(tm) Processor 8350, 2.0GHz, 16GB RAM

This project chooses the second approach because it gives the user a powerful way to transcribe his desired semantic field with several terms which came to his mind. On the systems side, it increases chances that the centre of the query terms actually lies in the desired semantic field. As averaging function the most simple one is chosen, the component-wise *arithmetic mean* (CAM) in the s-dimensional space:

$$\begin{aligned} \mathrm{CAM} &: \{u_j \in \mathbb{R}^s\} \longrightarrow \mathbb{R}^s, \\ \mathrm{CAM} &: \{u_1, \ldots, u_k\} \longmapsto \frac{1}{k} \sum_{j=1}^{k} u_k. \end{aligned}$$

Note that the arithmetic mean is based on the *Euclidean distance* metric. The problem is that the *cosine dissimilarity* is not a proper metric as discussed in Section 2.1. Hence it cannot be used to build an average. Since the query term vectors are normalized with respect to the L_1-norm (as all semantic vectors), the end of the vectors lie on the hyperplane that meet all axis at plus one. Geometrically the arithmetic mean can be interpreted as the mass centre of the endpoints on this plane and thus is justified to be used here.

Implementation Let n denote again the number of terms in the database. Starting at the resulting single centre c, the nearest neighbor search is done with a brute-force approach: for every term vector v_i, the distance $\mathrm{dis}(c, v_i)$ is calculated and the nearest ℓ vectors respectively their associated terms are returned. We denote these result terms with g_i for $i = 1, \ldots, \ell$. This idea could be implemented by first sorting all terms by distances $\mathrm{dis}(c, v_i)$ from smallest to largest and taking the first ℓ corresponding terms. This is of time complexity $\mathcal{O}(n \log n)$. To avoid a large memory footprint or the need of some external sort algorithm, the search is instead implemented with a *max-heap* where the ordering is with respect to $\mathrm{dis}(c, v_i)$. Iterating over all terms, the distance is calculated and put on the heap. The heap's size is bounded to ℓ, such that the furthest term gets removed in each step. In the end, the remaining ℓ terms on the heap are the nearest neighbors of the centre. Because it is a a *max-heap*, the removal is of constant time complexity. But the insertion in the heap has to seek an item to the bottom in worst case, and the heap depth is bounded to $\mathcal{O}(\log n)$. Doing n insertions, this again leads to $\mathcal{O}(n \log n)$ time complexity. So we did not improve with regard to time, but we now only need a single pass over the vectors v_i, and we need constant memory $\mathcal{O}(\ell)$.

Most algorithms that try to improve this search behind the $\mathcal{O}(n \log n)$-bound are not suitable for data with more than 20 dimensions as argued by Gionis, Indyk and Motwani [15]. Their paper present *Locality sensitive hashing* as one alternative to the brute-force algorithm that also works for more dimensions. It brings a speed improvement at the cost of accuracy and should be considered to make the search fast enough for a productive system.

Example 2.6. We start an example here and will develop it while extending the semantic relation extraction process. Table 2.1 shows the result list for the query terms "evolution" and "life".

Term	Distance
✠ evolution	0,178
darwinian	0,263
biology	0,314
biologists	0,338
darwinism	0,344
evolutionary	0,346
scientists	0,373
✠ life	0,377
selection	0,418
kuhn	0,424
reproduce	0,429
scientific	0,429

Table 2.1: The result list for the query "evolution life".

2.4.1 Clustering

In this section we explain how the search result from a query can be further analyzed. Consider the case when the related terms of "bug" are searched. "bug" denotes the animal but also an error in a computer program.[12] The two implied semantic fields are very loosely coupled. In linguistics such terms with different meanings but the same spelling and pronunciation are called *homonyms*.[13] They lead a problem. The semantic vectors of the query terms or at least some of them are very distant to each other. This can produce the following undesired results:

- The query vectors' average lies in a different semantic field than every query term.
- Some input terms are not respected at all, because their are overruled by multiple (more semantically related) query terms.

[12]The reason why a program error is called a "bug" is that in an early electromechanical computer a moth in the device was causing errors.

[13]If the terms have the same etymological origin one calls them polysemes, e.g. "read a *book*" vs. "*book* a hotel".

The system has to detect these cases and give appropriate feedback to the user. This is achieved by joining the set of all result terms and the query terms, and running a clustering algorithm on the joint sets. If the clustering algorithm classifies some or all query terms as outsiders, the system gives a warning message to the user with the so called problem terms P. This method is summarized in Algorithm 1. There are different clustering algorithms. The next paragraph explains why *DBSCAN* was chosen.

Algorithm 1 Analysis of quality of generated term list

Input: query terms q_j and thereof generated terms g_i
Output: set of weakly related query terms, called problem terms P

```
V = {ω(q_j)|j = 1,...,k} ∪ {ω(g_i)|i = 1,...,ℓ}
run cluster algorithm on V
P ← ∅                                          ▷ Problem terms
for all v_i ∈ V do
  if v_i is not part of any cluster then
    add term belonging to v_i to set P
  end if
end for
```

DBSCAN

The *DBSCAN* clustering algorithm [16] qualifies for our problem because of the following reasons:

- **No starting points required**: Beforehand we do not know how many clusters there are. The estimation of the number of clusters should be included in the clustering algorithm.
- **Any cluster shape is tolerated**: *DBSCAN* is able to identify clusters which are not of spherical nor elliptical form. This works because the search is based on density of points (not their distance). We need that property because the semantic fields can have any shape in the latent semantic space and field boundaries are in general not hard but identifiable by a decrease of density.
- **Detection of Noise**: *DBSCAN* can detect points not belonging to any cluster and classifies them as noise. We need this ability to determine if some query terms are semantically unrelated, which is the case when query term vectors are classified as noise.

These properties are visualized in Figure 2.3.

In the following we explain the idea of *DBSCAN* in more detail. *DBSCAN* takes two input parameters MinPts and ϵ. They are crucial for the following definitions.

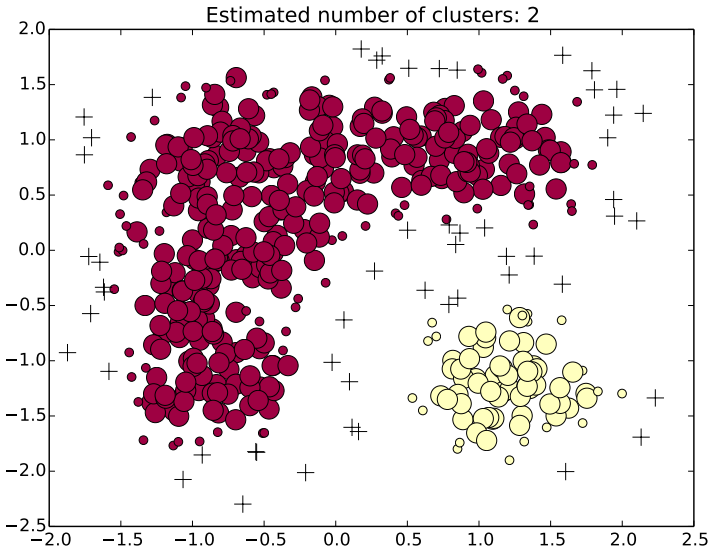

Figure 2.3: Application of the *DBSCAN* clustering algorithm to a set of demo points. Border points are drawn as small points and outsiders as crosses.

Definition 2.7. The *ϵ-neighborhood* of a point u, denoted by $N_\epsilon(u)$, is defined by

$$N_\epsilon(u) : \{v \in \mathcal{V} | \text{dist}(u, v) \leq \epsilon\},$$

where $\mathcal{V}$ is the set of all points and here dist is a proper distance metric[14] defined for their domain. Obviously the ϵ-neighborhood relation is symmetric, reflexive and not transitive.

The cluster algorithm first assigns each point into one of these classes (in this order):

- **Core points** are all points u that have more than MinPts within their ϵ-neighborhood, i.e. $|N_\epsilon(u)| \geq$ MinPts.
- **Border points** have less than MinPts points within their ϵ-neighborhood but are in the ϵ-neighborhood of a core point. Border points need a special treatment to find cluster boundaries (i.e. density drops), because the ϵ-neighborhood of border points in general contains significantly less points than the ϵ-neighborhood of a core point.
- **Noise points** are all points that are neither a core point nor a border point.

[14] Neighborhoods can also be defined more general for any dissimilarity measure.

Noise points already have their final label as outsiders and henceforth can be ignored. An edge is added between any two core points that have a distance smaller than ϵ, i.e. they are in each others ϵ-neighborhood (remember that the relation is symmetric). Each connected set of core points now results in a cluster. Finally each border point is arbitrarily assigned to one of the clusters containing its associated core points. This last step is the only non-deterministic one.

The run time complexity of this algorithm depends on the availability of efficient spacial access methods for region queries. If e.g. an $R^\star$-Tree with a worst case depth of $\mathcal{O}(\log n)$ is available, the run time complexity for a single region query is expected to be $\mathcal{O}(\log n)$.[15] The implementation proposed [16] uses at most one region query per point in the database, hence the total run time complexity is $\mathcal{O}(n \log n)$. But it degrades to $\mathcal{O}(n^2)$ if the spatial search for near points is done by brute-force.

Since *DBSCAN* relies on a metric, the *cosine similarity* disqualifies for the cluster analysis. In the system's prototype the semantic vectors are therefore interpreted in the Euclidean space. This approach has shown to work well, especially to detect unrelated query terms.

Example 2.8. We continue the Example 2.6 by adding the harmless looking query term "tree". The result list is shown in Table 2.2. For an non-computer scientist this result may be a surprise because the terms have totally changed. The reason lies in the added query term. "tree" is actually a polyseme, its structure in nature has motivated mathematicians to use it to describe an mathematical object. Since it is most often used in the context of computer science, and Wikipedia as our text corpus has a very high coverage of these fields, "tree" is strongly associated with this field. The query term "life" did not even find its way into the result list because it is too distant (0.72). Interesting is also that "parsimony" is the nearest term of the centre. If you know that parsimony is actually a method to estimate the structure of evolutionary trees of life [17], this result is reasonable.

[15] We additionally must assume that the ϵ-neighborhoods are small (i.e. have a small constant upper bound) compared to the total number of points.

Term	**Distance**	**Cluster**
parsimony	0,073	0
✠ tree	0,106	0
node	0,109	0
trees	0,116	0
nodes	0,117	0
huffman	0,126	0
bootstraps	0,127	0
grandparent	0,134	0
rebalancing	0,171	**-1**
bootstrap	0,279	**-1**
farris	0,354	**-1**
✠ evolution	0,629	**-1**

Table 2.2: The result list for the query "evolution life tree". Because "tree" is a polyseme and often used in the context of computer science, it is strongly associated with this field. It overrules the other query terms and produces a list with computer science terms.

Chapter 3

Collaborative Vocabulary Improvement

In Chapter 2 we saw how to extract semantic relations from Wikipedia and find the static representations of terms in a latent semantic space. We saw how to search for similar terms based on these representations and generate vocabulary lists out of one or several query terms. But there are several reasons why the result lists might not be satisfying:

1. The text corpus is not perfect. They may not or only weakly cover some semantic fields. Hence the extracted relations might not be satisfying.
2. There are terms that are simply not desired even if they belong to the desired field, because they are too abstract, too specific etc.
3. There are many plausible semantic fields that are not simultaneously representable in the semantic space alongside "stronger" semantic fields, e.g. music instruments (example in Appendix A).

All three cases are solvable with external help. A user should be able to adjust generated lists to his wishes. To solve the issue in the third case the user is forced to create the list completely by himself. What the system can contribute is that user input in the form of reviewed or self-created lists is conserved in time and made utilizable for other users. Of course this is already done by sharing whole lists with others, but it would be better to use the feedback to improve newly generated, *similar* lists too. In our system this goal is achieved by giving the user two options: A user can review a list and vote terms up or down or he can let the system *derive* the ratings for a particular list. Such a mechanism belongs to the field of *Recommender Systems*.

3.1 Content-based vs. Collaborative Recommender Systems

We denote a rating of an item z made by a user p with $r_p(z)$. When a rating is not explicitly made by the user but the system can estimate it based on other ratings, we speak about an estimated rating, denoted with $\widetilde{r}_p(z)$.

In a *content-based recommender system*, the rating $\widetilde{r}_p(z)$ is estimated based on previously explicitly given ratings of user p [18]. The system tries to find commonalities between the items by comparing their *static properties*. This properties only depend on the item and are equal for all users. For example a music recommendation system tries to identify interpreters, the style or origin of songs previously liked by p and derives by some heuristic if p likes a song he has not rated yet.

Collaborative Recommender Systems, also known as *Collaborative Filtering* requires several users. The rating $\widetilde{r}_p(z)$ is derived from ratings $\widetilde{r}_{p_j}(z)$ of *other* users p_j who are similar to p and rated item z too.[1] In contrast to content-based recommender systems, there are no static properties of the items used to derive ratings.

Our recommender system can be seen as a variant of a content-based one since we do not (or not yet) make a distinction between users. Indeed, our system is equivalent to a system where every action is done by a single anonymous user. We want the recommendations on the level of terms, hence items correspond to terms in our system. To derive a rating of a term in the context of a (newly generated) list L, we look at how the terms were rated in *similar* lists L_k. Hence we need a similarity measure between lists. This similarity measure serves as a weight of the influence another rating has. Our system contains also aspects from a collaborative recommender system, because the ratings of all users are (equally) used to improve the derivation of ratings.

Similarity Measure

There exist different measures of similarity. We will interpret lists as unordered sets of terms and use the following measure:

Definition 3.1. The normalized similarity between two sets (of terms) X and Y, known as *resemblance* or *Jaccard similarity*, is

$$\mathrm{sim}(X, Y) = \frac{|X \cap Y|}{|X \cup Y|} \in [0, 1]. \tag{3.1}$$

[1] Note that we now also need a measure of similarity between users. This measure can be based on separate user characteristics, but is usually calculated based on all the user's ratings.

3.2 Rating derivation

Let L denote some list. Let $L^{(c_j)}$ denote the cluster c_j of L, as it were found by the clustering algorithm described in Section 2.4.1. Note that L and c_j have both an identity apart from the terms they contain, i.e. multiple lists L_a, L_b or clusters c_i, c_j containing exactly the same terms can exist.

Before we can explain how ratings are derived, we have to care about a problem. Recall that lists may consist of clusters of terms which are from very different semantic fields, e.g. if the list is user-generated. An example is sketched in Figure 3.1. In this situation ratings from L_1 about terms in the intersection $L_1 \cap L_2$ are valuable to derive ratings in L_2. But ratings from L_2 have basically no weight because the similarity $\mathrm{sim}(L_1, L_2)$ is very low since L_1 has a second very big cluster not part of the intersection.

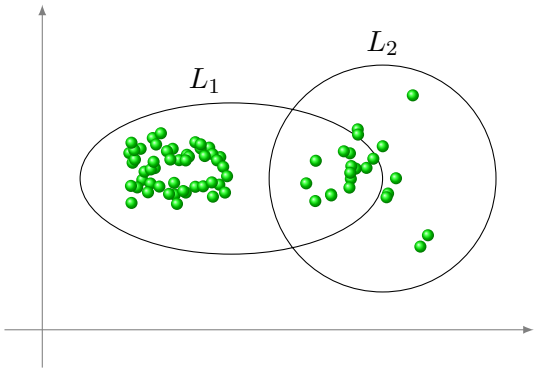

Figure 3.1: Two sample lists L_1 and L_2 in the latent semantic space showing the problem of clustered lists: Even though L_1 and L_2 have many common points in one cluster, their Jaccard similarity is very low.

The solution is to derive ratings based on similarities of clusters, instead of similarities of lists. Note that since Definition 3.1 is stated for sets, it also works for clusters. Hence $\mathrm{sim}(c_i, c_j)$ is valid to use. In this model, from each cluster c_i that a user reviewed, a rating can be retrieved. In the system's prototype a simple binary rating mechanism is used: the user can vote terms in a list up or down. We denote this user rating with

$$r_{c_i}(t) = \begin{cases} 0 & \text{if the user voted down term } t \text{ in the list } c_i \text{ belongs to,} \\ 1 & \text{if the user voted up term } t \text{ in the list } c_i \text{ belongs to.} \end{cases} \tag{3.2}$$

For all terms t (belonging to cluster c_j) of a list that has not been reviewed yet, we can derive ratings with the aggregation function [18]

$$\widetilde{r}_{c_j}(t) = K_{c_j} \sum_{c_i \neq c_j} r_{c_i}(t) \cdot \mathrm{sim}(c_i, c_j), \tag{3.3}$$

where K_{c_j} serves as a normalization factor and is given by

$$K_{c_j} = \frac{1}{\sum_{c_i \neq c_j} |\mathrm{sim}(c_i, c_j)|}. \tag{3.4}$$

As we see from the formula, $\mathrm{sim}(c_i, c_j)$ serves as the weights for the aggregated ratings and is equal to zero if c_i and c_j do not share any terms. In the system's prototype the estimated rating is interpreted like that: For a threshold value $R = 0.5$, if $\widetilde{r}_{c_j}(t) >= R$ for the cluster c_j to which t belongs to, then t is *active*, meaning it is displayed in the list. Otherwise t is proposed to be removed from the list. See Section 4.2 for more details how ratings could be interpreted.

An open question remains: when some terms are voted down respectively are removed from the list, should they still be considered for similarity calculations? They should not, which becomes clear in the example sketched in Figure 3.2. Because a user removed nearly all terms from L_2 that are in the intersection with L_1, L_1 should influence the derived ratings in L_2 very little. This leads to an update of the Definition 3.1:

Definition 3.2. The normalized similarity between two clusters c_i and c_j is

$$\mathrm{sim}'(c_i, c_j) = \mathrm{sim}(\{t \in c_i | r_{c_i}(t) > R\}, \{t \in c_j | r_{c_j}(t) > R\}), \tag{3.5}$$

where R is again a threshold value to set a term as *active*.

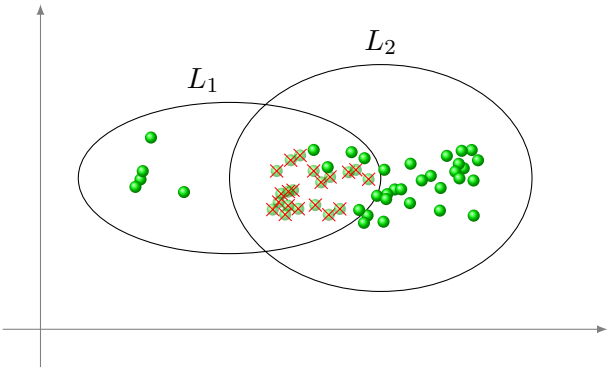

Figure 3.2: Two sample lists L_1 and L_2 in the latent semantic space where L_2 deleted most of the terms (drawn crossed out) from the intersection with L_1.

Example 3.3. In this example we first generate the two lists shown in Table 3.1 and then generate a third list where we let the system derive the ratings (Table 3.2).

Term	Rating
✠ traffic	1,00
taxis	1,00
commuter	1,00
connecting	1,00
congestion	1,00
⊘ *rail*	0,00
⊘ *railway*	0,00
trams	1,00
⊘ *trains*	0,00
expressway	1,00
transport	1,00
routes	1,00

(a) "traffic"

Term	Rating
✠ train	1,00
trains	1,00
rail	1,00
commuter	1,00
⊘ *roads*	0,00
railway	1,00
freight	1,00
fares	1,00
tunnel	1,00
monorail	1,00
railways	1,00
eurotunnel	1,00

(b) "train"

Table 3.1: The result lists for the queries "traffic" and "train". Terms marked with ⊘ were voted down by the user and thus have a rating of zero.

Term	Rating
✠ taxi	1,00
taxis	1,00
commuter	1,00
minibus	1,00
trams	1,00
road	1,00
fares	1,00
expressway	1,00
railway	0,54
trains	0,54
rail	0,54
roads	0,00

Table 3.2: The result list for the query "taxi". The system derived the ratings based on the user ratings given in Table 3.1a and Table 3.1b. The terms "railway", "trains" and "rail" have a middle rating, because they were only voted down in the "traffic"-list but appear also in the "train"-list.

Chapter 4

System Prototype

For demonstration purposes, a browser-based prototype of the trainer was created. It is available under demo.vocminder.com. The back-end is built with *Python*[1] and *Django*[2]. As front-end the responsive *Bootstrap*[3] template system was choosen.

4.1 Data Model

We will have a look at the most important model components. The full data model of the trainer can be found in Appendix B. With Django, models are described with Python classes and are then mapped to a relational model (MySQL[4]) automatically. This results in the following (simplified) relations. Primary keys are bold, foreign keys are underlined.

- List (**list_id**, name, ...)
- Term (**term_id**, language_id, text, data)
- Occurrence (**occurrence_id**, list_id, term_id, is_query, cluster, distance, rating)

The text attribute of the Term relation holds the term itself and data is a Binary Field into which the semantic vector is serialized. When a query is executed, a new List table is created and each returned term is stored as an entry in a Many-to-Many-Relation between List and Term in the Occurrence table. Its associated attributes are self-explaining and hold all the information gained through the semantic relation extraction described in Chapter 2 and the rating mechanism described in Chapter 3.

[1] See www.python.org
[2] See www.djangoproject.com
[3] See getbootstrap.com
[4] See dev.mysql.com

4.2 User Interaction

In the following we list some use cases that are implemented in the prototype. Each use case is roughly associated with one view accessible from the menu bar.

- The **Generate View** allows the user to enter arbitrary query terms and the desired length ℓ of the vocabulary list. If the query terms are found in the Term relation, the system is doing the semantic nearest neighbor search and the clustering. The resulting list presents each term with the distance to the query term centre and the cluster index, where minus one means that the term is classified as an outsider. If some or all query terms are not included in any cluster, a warning is displayed that the query terms are difficult to relate. Figure 4.1 shows a screenshot of this view.
- In the **List Overview** the user can browse through generated lists, create new lists or delete existing ones.
- In the **Detail View** of a list, the user can vote terms up or down within the list which make them display *enabled* or *disabled* respectively. Alternatively, the user can let the system derive the ratings. The idea is to combine this two mechanisms. The user lets the system derive the initial ratings and adjusts them where needed. This can also be seen in Figure 4.1.

4.3 Possible Extensions

The following components not implemented in the prototype could be included in a productive system:

- Multi-user support: In the prototype we only considered one anonymous user. In a productive system we should differentiate between users in at least two areas:
  - Vocabulary lists should be controlled by their creators, hence access and modifications should be monitored. A more automatic solution for the recommender system component would be to assess the users' expert knowledge and use this as a weight for the impact of their ratings.
  - The recommender system could be extended to also use collaborative filtering methods, to reflect different desires of users. Primary school students may want vocabulary lists of simple terms while students prefer lists of an extended range of terms.

- For generate queries, the user should be able to dynamically extend the generated list by more terms. This could be done by simply searching further around the query terms centre, or use approved result terms as the new set of query terms, since their centre will more probably lie in the desired semantic field in the latent space.
- An integrated training system: After the terms are automatically translated, the user could train them with different training schemes. A lot more extensions are imaginable to improve the trainer component, such as learning progress statistics and progress-sensitive training schemes.

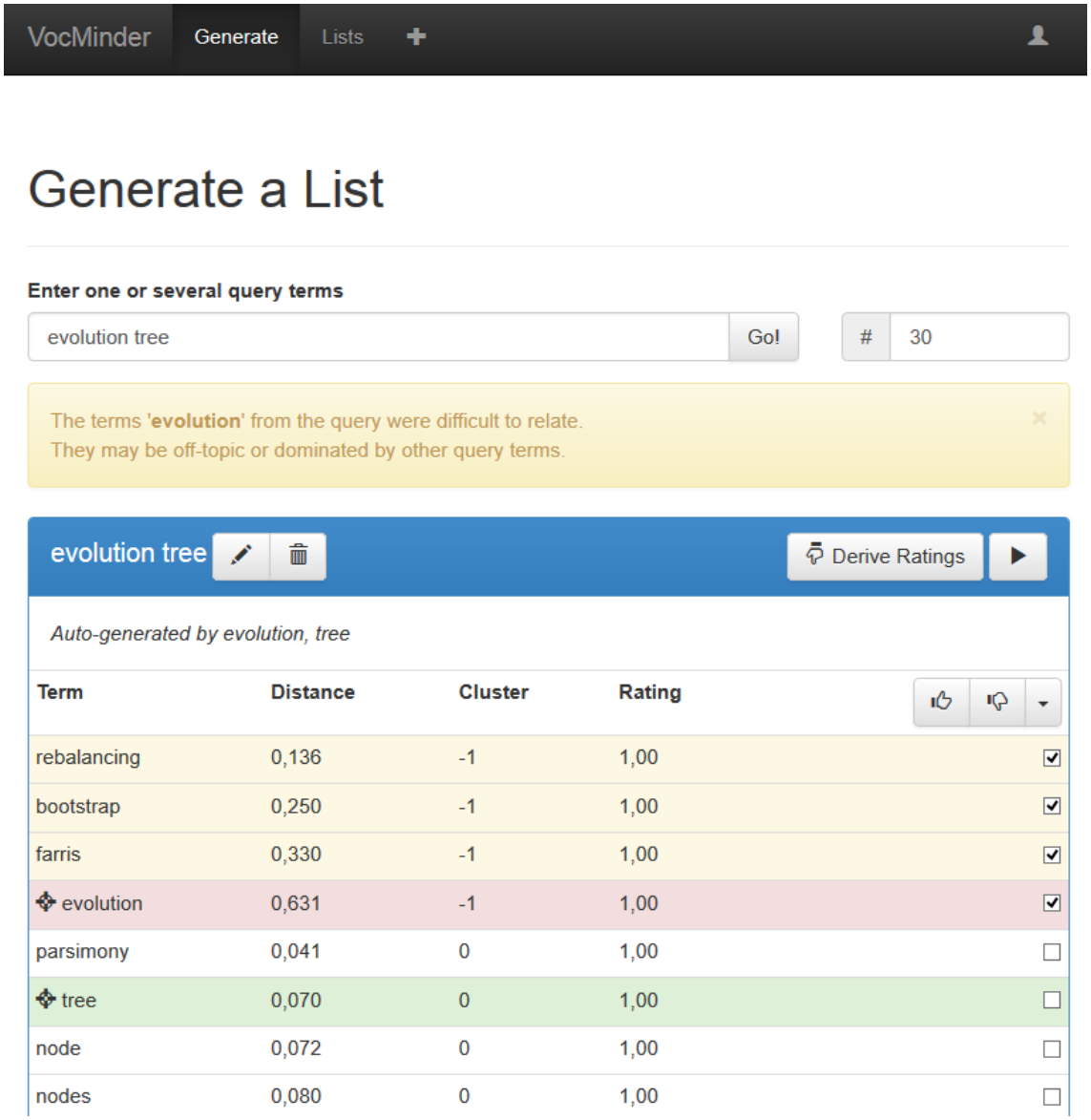

Figure 4.1: A screenshot of the **Generate View** where the user can enter query terms.

Chapter 5

Conclusion and Outlook

We presented the prototype of a system that combines several techniques from information retrieval and analysis and combines it with a recommender system. While PLSA is a well-known tool for automatic semantic relation extraction, we showed how it is applicable to a large knowledge base. Given this base in any desired language, our system allows to relate terms of a large-size vocabulary in a semantic space model. From this model it is possible to generate vocabulary lists for the user's needs, by just naming some sample terms from the desired field. Even though the semantic model is static, it can steadily improve itself by using everyone's feedback.

While all the components work like they were intended, the overall system is not advanced enough to be used in practice yet. However, the system can be seen as an experiment that combines well-known techniques from different areas and gives an idea what can be built upon.

The semantic relation extraction does not use the whole Wikipedia, thus it is difficult to predict for which semantic fields the extracted semantic vectors are meaningful. To be able to extend the existent system to use all Wikipedia articles available, a parallel version for PLSA or an incremental variant would have to be used. The quality of the semantic relations could then be properly evaluated (e.g. by comparing them with manual judgment). Generally an automatic estimator of the quality of semantic relations should be added. When the quality is low, the user should be informed.

As we have seen in the Related Work Section 1.1, there are other options for the latent semantic analysis part. Especially Wikipedia-based *Explicit Latent Semantic Analysis* [8] should be explored because it additionally uses the Wikipedia categories as structure information, which could lead to better results.

Another way to improve the relation extraction is to use stemming in the preprocessing step. The stemming algorithm does reduce the terms to its stems and PLSA is then run on these reduced terms [9]. Note however, that we would have to program the stemming algorithm carefully and separate for every single language to be supported.

For a productive system we would have to increase the responsiveness of a generate query. Semantic relations could be explicitly stored for near terms to speed up the search to constant time complexity at the cost of a larger database.

The recommender system part can be extended in many ways. A training component with user profiles storing their training progresses would lead to many additional possibilities to improve the relatedness information. For example a user's training progress could be used as implicit feedback on the quality of individual terms or their relatedness.

Bibliography

[1] Nation, I., Newton, J.: Teaching vocabulary. Heinle, Cengage Learning (1997)

[2] Hsu, M.H., Tsai, M.F., Chen, H.H.: Query expansion with conceptnet and wordnet: An intrinsic comparison. In: Information Retrieval Technology. Springer (2006) 1–13

[3] Tikk, D., Yang, J.D., Bang, S.L.: Hierarchical text categorization using fuzzy relational thesaurus. Kybernetika **39**(5) (2003) 583–600

[4] Panchenko, A., Adeykin, S., Romanov, A., Romanov, P.: Extraction of semantic relations between concepts with knn algorithms on wikipedia. In: Proceedings of Concept Discovery in Unstructured Data Workshop (CDUD) of International Conference On Formal Concept Analysis. (2012) 78–88

[5] Navigli, R., Ponzetto, S.P.: Babelnet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. Artif. Intell **193** (2012) 217–250

[6] Miller, G.A.: Wordnet: a lexical database for english. Communications of the ACM **38**(11) (1995) 39–41

[7] Hofmann, T.: Probabilistic latent semantic analysis (January 23 2013) Comment: Appears in Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI1999).

[8] Gabrilovich, E., Markovitch, S.: Computing semantic relatedness using wikipedia-based explicit semantic analysis. In Veloso, M.M., ed.: IJCAI. (2007) 1606–1611

[9] Takale, S.A., Nandgaonkar, S.S.: Measuring semantic similarity between words using web documents. International Journal of Advanced Computer Science and Applications(IJACSA) **1**(4) (2010)

[10] Panchenko, A., Morozova, O.: A study of hybrid similarity measures for semantic relation extraction. In: Proceedings of the Workshop on Innovative Hybrid Approaches to the Processing of Textual Data, Association for Computational Linguistics (2012) 10–18

[11] Pucher, M.: Wordnet-based semantic relatedness measures in automatic speech recognition for meetings. In: Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions, Association for Computational Linguistics (2007) 129–132

[12] Heilman, M., Eskenazi, M.: Application of automatic thesaurus extraction for computer generation of vocabulary questions (December 04 2008)

[13] Hajian, B., White, T.: Measuring semantic similarity using a multi-tree model. In: Workshop chairs. (2011)

[14] Strube, M., Ponzetto, S.P.: Wikirelate! computing semantic relatedness using wikipedia. In: Proceedings of the Twenty-first National Conference on Artificial Intelligence, AAAI Press (2006)

[15] Gionis, Indyk, Motwani: Similarity search in high dimensions via hashing. In: VLDB: International Conference on Very Large Data Bases, Morgan Kaufmann Publishers (1999)

[16] Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In Simoudis, E., Han, J., Fayyad, U., eds.: Second International Conference on Knowledge Discovery and Data Mining, Portland, Oregon, AAAI Press (1996) 226–231

[17] Gonnet, G., Scholl, R.: Scientific Computation. Cambridge University Press, Cambridge, England, UK (2009)

[18] Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Trans. Knowl. Data Eng **17**(6) (2005) 734–749

Appendix A

Sample List

Term	Distance	Cluster
orchestras	0,031	0
virtuoso	0,032	0
trumpets	0,073	0
violins	0,074	0
orchestral	0,075	0
serenade	0,078	0
sopranos	0,079	0
timpani	0,083	0
soloist	0,084	0
violas	0,085	0
✠ cello	0,086	0
✠ violin	0,100	0

Table A.1: The result list for the query "violin cello". We see that the semantic analysis does not produce a tight relation between music instruments because they are used in other contexts in the corpus.

Appendix B

The Django Model

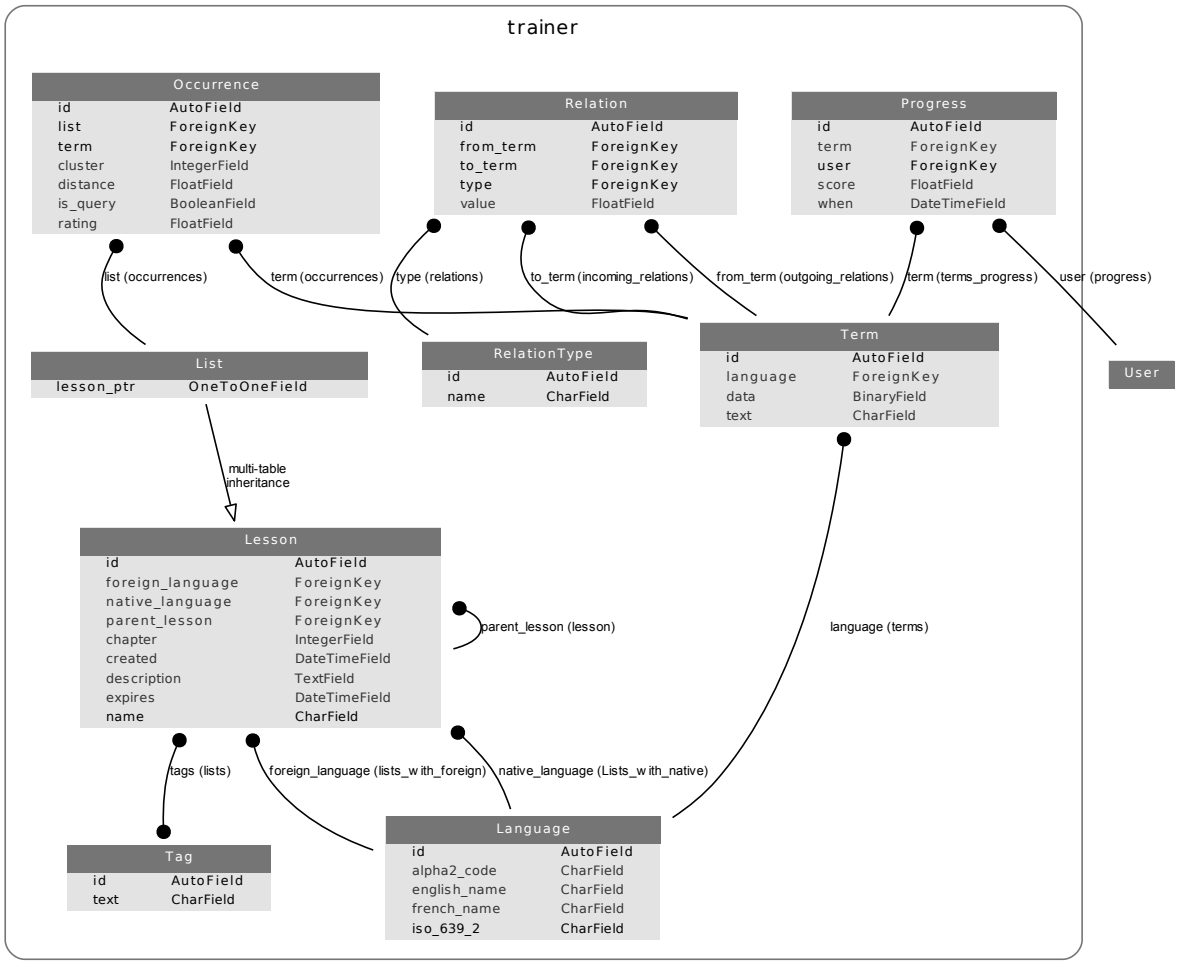

Figure B.1: The model of the trainer.