Universitat Oberta de Catalunya (UOC)

Máster Universitario en Ciencia de Datos (*Data Science*)

Master's degree thesis

Area: NLP & Text Mining from Wikipedia/DBPedia

Automatic Query Expansion for Vehicle Repair Documents through User Behavior

NLP, embeddings, synonyms, query expansion

Author: Juan Carlos Ghiringhelli

Tutor: Nadjet Bouayad-Agha

Professor: Oriol Collell Martin

Barcelona, June 24, 2020

Copyright

Master's degree thesis

Title:	Automatic Query Expansion for Vehicle Repair Documents through User Behavior
Author:	Juan Carlos Ghiringhelli Jueguen
Tutor & Collaborator:	Nadjet Bouayad-Agha
Practicioner:	Oriol Collell Martin
Turn in date:	06/2020
Degree:	Data Science
Thesis field:	NLP & Text Mining from Wikipedia/DBPedia
Language:	English
Keywords	NLP, embeddings, synonyms, query expansion

Dedicated

To my kids, my father, and my wonderful wife Marcela who makes it all worth it.

Thanks

To Nadjet who was of great help during the journey, Juan Alfonso & Juan Pedro who made it possible to live in beautiful Catalunya, my co-workers Aaron, Bill, Bob, Curt, Greg and Thou.

Abstract

The process of information retrieval Information Retrieval (IR) by query driven search engines have become an essential part of the customer experience in any data related digital product. The accuracy and completeness of the search results is a matter of great interest and a crucial key performance indicator. An important enhancer for search engines is query expansion Query Expansion (QE), where equivalent search queries Equivalent Search Query (ESQ) are added to the original request to increase recall. ESQs can be discovered using the same tools as synonym discovery given certain considerations, taking advantage of the fact that synonym discovery is a well developed field of Natural Language Processing (NLP) with many available techniques.

The motivation for this project is to use the tools available in NLP Machine Learning (ML) to automatically detect ESQs. For this a large sample of logs describing search query customer behavior was used. This data set was obtained from a live enterprise product that publishes repair documents for automobiles. Graph embeddings through an implementation method called node2Vec and vector cosine similarity is the chosen discovery method for the ESQs.

The conclusion of the experiment is that while usable search expansion queries are discovered, extra human intervention or further automatic selection is necessary to filter the valuable cases from the large number of found cases, even working within a strict similarity threshold.

El proceso de recuperación de información IR de los sistemas de búsqueda centrados en consultas se ha vuelto un elemento fundamental para la experiencia del cliente en cualquier producto que almacene datos digitales. La exactitud y completitud de los resultados resulta de gran interés y es uno de los principales indicadores clave del negocio. Expandir la consulta original con términos relacionados es una potente herramienta para incrementar el retorno. Aunqué con ciertas consideraciones, las consultas de búsqueda equivalentes pueden ser detectadas utilizando las mismas herramientas que para el descubrimiento de sinónimos, algo que resulta provechoso dado lo prolífico que es este campo del 'Natural Language Processing' (NLP por sus siglas en inglés).

La motivación princial del proyecto es la detección de estas consultas por medio de las herramientas propias del NLP. Para esto se utilizó un gran conjunto de datos detallando el comportamiento de usuarios al utilizar sistemas de bésqueda. El conjunto de datos fue extraido

del sistema de búsqueda de un producto empresarial que publica documentos de reparación de automoviles. Se seleccionó el método conocido como node2Vec donde se generan grafos embebidos sumada a la similitud dada por el coseno de los vectores para hallar las consultas equivalentes.

La conclusión del experimento es que si bien se encontraron consultas equivalentes útiles aún es necesario una intervención de expertos o mejorar la selectividad para una mejor discriminación de las consultas valiosas dentro del gran número de equivalencias encontradas, incluso al utilizar un umbral muy estricto de similitud.

Keywords: NLP, embeddings, synonyms, query expansion

Contents

Abstract ix

Index xi

Figures index xv

Tables index xvii

1 Introduction 3

1.1 Project Summary . . . 3

1.2 Current state of the project . . . 4

1.2.1 Bags-of-Words and N-Grams . . . 5

1.2.2 Full-text search . . . 6

1.2.2.1 Index creation . . . 6

1.2.2.2 Search . . . 6

1.2.2.3 Configuration . . . 6

1.2.2.4 Order of display . . . 7

1.2.3 Misspellings and Caps . . . 7

1.3 Quality metrics . . . 8

1.3.1 Precision . . . 8

1.3.2 Recall . . . 8

1.3.3 Accuracy . . . 9

1.3.4 F1-score . . . 9

1.4 Personal motives . . . 9

1.5 Objectives . . . 9

1.6 Methodology . . . 10

1.7 Project planification . . . 11

2 State of the art **13**
2.1 Introduction . . . 13
2.2 History of NLP . . . 13
2.2.1 Proto NLP . . . 14
2.2.2 The 1950s . . . 15
2.2.3 First chatterboxes and ontologies - 1960s & 1970s . . . 16
2.2.3.1 Early ontology work . . . 16
2.2.3.2 Chatterbots . . . 17
2.2.4 Machine learning and supervised methods - 1980s & 1990s . . . 17
2.2.5 Neural Networks and unsupervised methods - XXI Century . . . 18
2.2.5.1 Self-supervised learning . . . 18
2.3 Synonyms and term equivalency . . . 18
2.3.1 Synonyms related research . . . 20
2.3.1.1 Historical User Data . . . 20
2.3.1.2 Linguistic resources . . . 20
2.3.1.3 Lexical Synonyms . . . 20
2.3.1.4 Word embeddings . . . 21
2.4 Lexical semantics . . . 21
2.4.1 Distributional semantics . . . 21
2.4.1.1 Vector semantics . . . 22
2.4.2 Word embeddings . . . 23
2.4.2.1 Word Embedding Techniques . . . 23
2.4.2.2 Sparse matrices - LSA . . . 24
2.4.2.3 Dense matrices - Word2Vec . . . 25
2.4.2.4 Embedding layer . . . 27
2.4.2.5 One-Hot encoding . . . 27
2.5 Data sanitation and preprocessing . . . 28
2.5.1 Stemming . . . 28
2.5.2 Lemmatization . . . 28
2.5.3 Morphological segmentation . . . 29
2.6 Neural networks . . . 29
2.6.1 Brief history of Neural Networks . . . 31
2.7 Query Expansion . . . 31
2.7.1 The Vocabulary Problem . . . 32
2.7.2 Query Expansion Workflow . . . 34
2.7.2.1 Data preprocessing . . . 34

2.7.2.2 Term extraction . . . 35
2.7.2.3 Term Weights and Ranking . . . 35
2.7.2.4 Term Selection . . . 35
2.7.2.5 Query Reformulation . . . 36
2.7.3 Brief History of Query Expansion . . . 36
2.7.3.1 Research and preparation . . . 36

3 Data Acquisition and Preprocessing 39
3.1 Diagnostic Troubleshooting Codes . . . 39
3.2 Document queries . . . 40
3.2.1 Acquisition - Splunk logs . . . 40
3.2.2 Preprocessing . . . 40
3.3 SMEs Created Synonyms . . . 41
3.3.1 Acquisition - SQL Schema . . . 42
3.3.2 Preprocessing . . . 42

4 Term Weight, Ranking and Selection 43
4.1 Term Weighting & Ranking - Graph embeddings . . . 43
4.1.1 Graph embeddings - Node2Vec . . . 44
4.1.2 Model hyper-parameters . . . 45
4.1.2.1 Grid search . . . 46
4.1.2.2 Parameter visualization . . . 47
4.1.3 Model Training . . . 49
4.1.3.1 Sentence Embeddings for Out-of-Vocabulary . . . 50
4.1.3.2 OEM Specific Terms Discussion . . . 51
4.2 Term Selection . . . 51
4.2.1 Softmax . . . 52

5 Evaluation & Metrics 53
5.1 Evaluation . . . 53
5.1.1 Subject Matter Expert Equivalence Ranking . . . 53
5.1.2 Subject Matter Expert Feedback . . . 55
5.2 Metrics . . . 57

6 Conclusions & Future Work 59
6.1 Conclusions . . . 59
6.2 Future Work . . . 60

Bibliography 60

List of Figures

1.1 Bag of words schema 5
1.2 N-gram schema 5
1.3 Full-text search schema with 1-grams 6

2.1 Volvelle - Generation Reborn and Reformed (Part II) - Reproduction 14
2.2 Vector Semantics. Source: Speech and Language Processing (3rd Edition) 22
2.3 Word Mover's Embedding: Universal Text Embedding from Word2Vec 23
2.4 Word2Vec 25
2.5 Skip-gram 26
2.6 One-Hot encoding 28
2.7 Neural Network. Source: Understanding Neural Networks, Tony Yio 30

3.1 DTC Codes 39

4.1 Node2Vec sentence representation 45
4.2 Node2Vec 45
4.3 Node2Vec Parameters 46
4.4 Node2Vec DFS - BFS 46
4.5 Hyper-parameter visualization 49
4.6 Softmax regression 52

5.1 Equivalent Search Queries Matching Ranking 54
5.2 2014 Chevrolet Silverado 1500LT 5.3L 56

List of Tables

2.1 Vocabulary problem methods. 33
2.2 Alternative problem methods. 37

3.1 Splunk Logs List [Search Query - Document Id] 41
3.2 Subject Matter Experts' hand curated synonyms 42

4.1 Node2Vec hyper-parameter ranges for optimization 47

5.1 Subject Matter Experts' Ranked ESQ Summary 54
5.2 Examples of found ESQ by type 55
5.3 Search queries retrieved documents 57

Glossary

ACES Aftermarket Catalog Exchange Standard. 4

ANN Artificial Neural Networks. 29–31

BoW Bags-of-words. 4, 5, 24, 26, 60

CBOW Continuous Bag-of-Words. 25, 26

DTC Diagnostic Troubleshooting Codes. 39

ESQ Equivalent Search Query. ix, 3, 4, 7–10, 19, 25, 32, 34, 41, 42, 46, 50–60

FTS Full-Text Search. 4–7, 32, 57, 58

HTML Hyper Text Markup Language. 4, 48

IR Information Retrieval. ix, 31, 32

LSA Latent Semantic Analysis. 20, 24, 27

MAP Mean Average Precision. 58

ML Machine Learning. ix, 4, 27, 34, 35, 54, 59, 60

NLP Natural Language Processing. ix, 4, 13–18, 21, 23, 27–29, 31, 35, 36, 60

NN Neural Networks. 18, 23, 25, 27, 29–31

OCR Optical Character Recognition. 5, 7

OEM Original Equipment Manufacturers. 3–5, 51, 55, 56

OOV Out-of-Vocabulary. 10, 50, 51

PDF Portable Document Format. 4

QA Quality Assurance. 10

QE Query Expansion. ix, 4, 7, 8, 31–34, 36, 37, 39, 56

RW Random Walks. 44, 45, 48, 49

SGNS Skip-gram with Negative Sampling. 25, 26

SME Subject Matter Experts. 3, 4, 7, 8, 10, 41, 42, 46, 52–56, 58, 59

SQ Search Queries. 3, 7, 19, 37, 40–44, 48, 50–53, 57, 58, 60

SQL Structured Query Language. 3, 4, 40, 42, 57

TF-IDF Term-Frequency / Inverted Document Frequency. 24, 26, 51

VRD Vehicle Repair Documents. 3, 4, 9, 37

XML eXtensible Markup Language. 32, 36

Chapter 1

Introduction

1.1 Project Summary

Nowadays in the information era a robust and powerful query-driven search engine is expected by most users of digital data repositories. Users expect both a high rate of recall, meaning the documents they expect are retrieved, and precision, meaning those documents won't be buried under a large amount of irrelevant documents by showing the relevant ones at the first positions. This field is a major area of interest and has grown significantly over the last few years both in scope and complexity. Query expansion by usage of equivalent queries is one of many ways to improve on the underlying search mechanism. Due to natural language complexity, specific domain terms, and differences in word context and languages, discovery of these equivalencies is not a simple task.

At my current workplace, an automobile related data company, the flagship product provides to the customers different types of Vehicle Repair Documents (VRD) that help mechanics perform repair and maintenance on the vehicles. The system is a typical front-end website fed by a back-end service that runs Search Queries (SQ) on an indexed Structured Query Language (SQL) database. Given the different vehicle manufacturers, called in the vehicle world Original Equipment Manufacturers (OEM), it is a very common occurrence that specific parts of the car are referred by different words, i.e.: the term "rear bumpers" might be referred to as "rear fenders", "rear guards", "back bumpers", etc. These different words with similar meaning are formally synonyms and while they do not always share the exact same meaning, in this specific context they usually do.

In the current search engine system, query expansion is used but ESQs creation is a time consuming process that involves Subject Matter Experts (SME)s finding them by way of manual research and personal knowledge. For this purpose a very simple tool that does no other tasks other than the typical Create, Read, Update, Delete *(CRUD)* operations is used. Using the

metadata and content from each document, an asynchronous process generates a single search index using SQL Full-Text Search (FTS). This process uses Bags-of-words (BoW) [1.2.1] for each document. This index is used by the search engine to retrieve the documents.

An automatic ESQ discovery system can be created by leveraging the capabilities of NLP and ML. Data pertaining to the behavior of the users who in turn are SMEs can be used for this purpose. It is worth mentioning that in the vehicle repair domain and the specific flagship product from which we will draw the data and test the results, all the searches are done in the context of a specific vehicle, e.g.: 2003 Toyota Camry LE 3.0L. These vehicles are defined by a standard called Aftermarket Catalog Exchange Standard (ACES) [1]. The most frequent case is for the user to be looking in the context of one of these vehicles for a very specific document instead of related different things out of amusement, curiosity or convenience.

Such a system would greatly reduce the effort needed to maintain the ESQs creation and potentially give new unforeseen insights into the problem. Therefore, the company has identified inaccurate and incomplete search results as one of the main reasons why customers unsubscribe from the flagship product, and considers the use of QE as a major contributor to the improvement on search results. Product leadership has shown great interest in this project.

It is important to note that while synonym discovery is an important task in NLP and strongly related to query expansion, it does not share the same objectives so it cannot be treated as identical. Synonym discovery deals with similarity of word meaning in the context of natural language, while query expansion is only interested in the recalled documents in a search engine as standalone search query requests. How similar are those two tasks depends on the underlying specific implementation of the search engine system, and this similarity is hard to assess. Since it is important for the querying system to consider the synonyms for the accuracy of the user experience as most mechanics do not know or remember the specific terms each OEM use for each model and year, we use synonym discovery techniques as our starting point.

1.2 Current state of the project

First I will explain some basic concepts used in search and indexing like n-grams and BoWs, followed by some relevant features of the current search engine.

We will describe the current architecture of the search engine. This engine indexes VRDs composed of words written in the English language and is contained in different formats, normally Hyper Text Markup Language (HTML) and Portable Document Format (PDF). Both

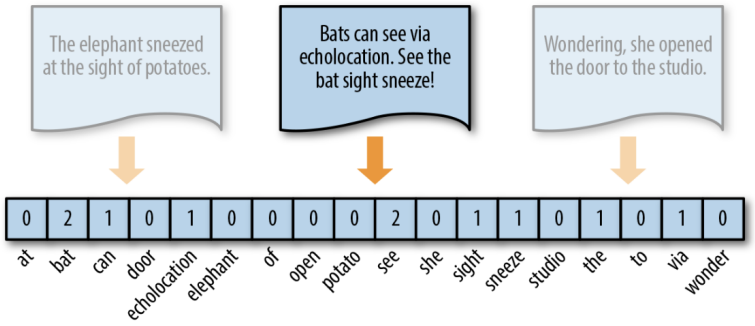
[1] https://www.autocare.org/uploadedfiles/autocareorg/what_we_do/technology/resource_files/what_aces_and_pies_is_not.pdf ACES

text extraction techniques and Optical Character Recognition (OCR) are used to populate the BoW [1.2.1], which are then used to create the FTS index.

Both third-party content in the form of official repair documents published by the OEMs and first-party content where the result of a hot-line assistance phone call is captured as an archive are included. This results in a combination of both formal and informal language, where *Chevrolet* might be referred as *Chevy*.

1.2.1 Bags-of-Words and N-Grams

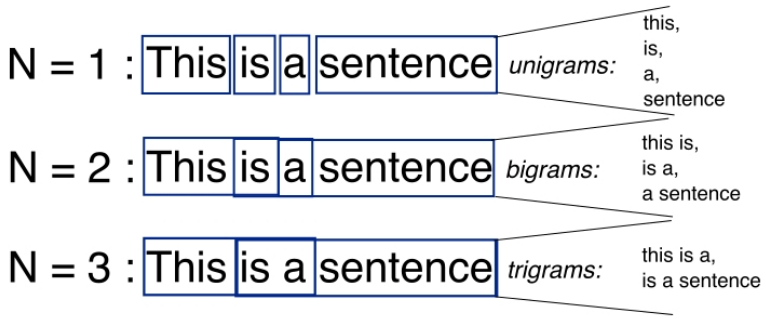
⟨bow⟩ Bags of words are sets of unique terms contained in the documents that keep track of the frequency of appearance but ignore order, grammar and context. They are commonly used for document classification like spam detection and, as in this case, search mechanisms.



Source: https://towardsdatascience.com/from-word-embeddings-to-pretrained-language-models-a-new-age-in-nlp-part-1-7ed0c7f3dfc5

Figure 1.1: Bag of words schema

An n-gram is a division of a string into sequences of n words or n characters.



Source: https://www.oreilly.com/library/view/fasttext-quick-start/9781789130997/8ddb990b-18a3-4e09-8d63-c38f10965055.xhtml

Figure 1.2: N-gram schema

1.2.2 Full-text search

FTS is an indexing method that has two well defined steps, index creation and search.

1.2.2.1 Index creation

For the complete set of documents of the system, a vocabulary is created with all found n-grams, excluding stop words such as articles and pronouns. A single index, also called "concordance", is created to quickly find which documents contain the search terms.

1.2.2.2 Search

After the index is ready then an inverted tree is created where each term has knowledge of which documents contains the term itself.

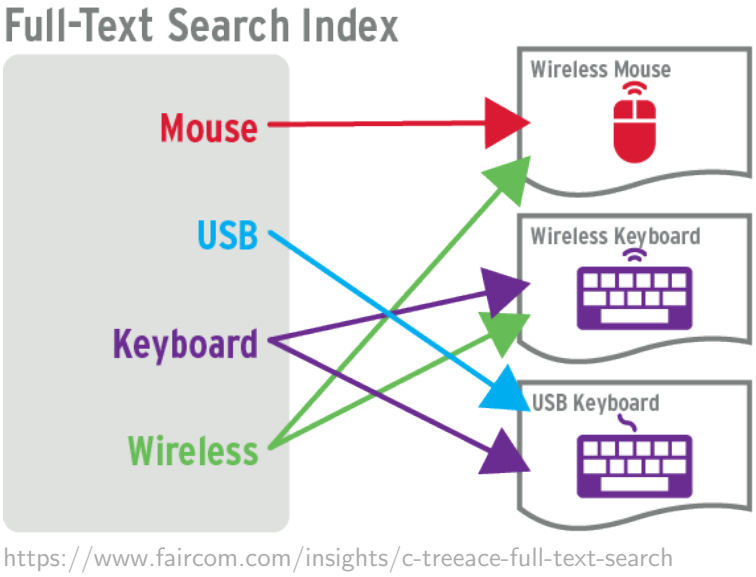


Figure 1.3: Full-text search schema with 1-grams

1.2.2.3 Configuration

Some of the important configurations for this particular FTS system are:

1. For performance reasons, the default minimum amount of letters for each term of three is used. The more indexing that needs to be done, the slower the document retrieval and the more expensive the index building process is.

2. Words are being kept in their original format, no reduction or modification is being done [2.5].

3. Caps are not ignored which greatly impacts recall. This was an early bad decision and while it made sense at the time is not justifiable in hindsight.

Any change on these parameters implies rebuilding the whole index from scratch which takes weeks given the sheer amount of data involved, about 40 million documents, each one with a body of text, potential several images to OCR and document metadata.

1.2.2.4 Order of display

⟨order⟩ Though a ranking score is calculated when retrieving the documents for each recalled document and SQ, FTS has no stance on the order of the recalled documents. The current system does not use ranking, opting instead for a combination of two other different criteria, namely document taxonomy and count of words in the title.

Document taxonomy is a categorization system for each document that was created by the company's SMEs. Some of these categories are electrical wiring diagrams, remove & replace, diagnostic troubleshooting codes, etc.

Word count in title is a simple sum of how many of the search terms appear in the document title. Titles are given as metadata of the document and not necessarily appear completely or partially in the document's body.

Recalled documents are shown first by alphabetical ordered category, and second by word count in title. This system has proved to be troublesome at best according to the product team research, and while the improvement of ESQs can have positive side effects, improving it is outside the scope of this project.

1.2.3 Misspellings and Caps

In the current system:

1. No automatic correction of misspelled words exists.

2. No dictionary of automotive terms is used for correction.

3. As we mentioned before, search is case sensitive.

This means Rear, Reer, reer, reaR, REAR, rear, rer, reat, etc., are all considered different words. If a search done with any of those terms returns no documents, the system does nothing with the terms themselves to improve recall, it will rely on the SMEs' created ESQs to potentially bring more documents through QE.

When a certain SQ that retrieves no documents is looked upon a significant amount of times, the information is passed to the SMEs who then add the most common cases by hand,

including orthographic errors (e.g., utilise and utilize) and lexical variations (e.g., use vs utilize). This is an asynchronous, human resource-bound and hard to maintain partial solution. Even while a much better solution would be to add a dictionary and use a word distance method for correction, the current project has some potential to alleviate this problem by automatic discovery of misspellings and morphologies as ESQs.

1.3 Quality metrics

In this section the metrics typically used for QE, including the important and widely referred concepts of recall and precision, will be explained. As we will see in chapter 5, it was actually unfeasible to directly calculate these metrics due to the system characteristics. The explanation is still included due to the importance and constant use of the terms in the document.

Given a confusion matrix, with false/true negatives/positives, we can define four metrics.

1.3.1 Precision

Precision is the measurement of correctly identified positive cases over all the predicted positive cases. A high score means we have few false positives. For our system, precision is how many relevant documents were returned over all returned documents:

$$\frac{returned\_relevant\_documents}{returned\_relevant\_documents + returned\_irrelevant\_documents}$$

.

1.3.2 Recall

Recall is the measurement of correctly identified positive cases over all the positive cases. A high score means we have few false negatives. For our system, recall is how many relevant results were returned over total number of relevant results in the DB:

$$\frac{returned\_relevant\_documents}{returned\_relevant\_documents + non - returned\_relevant\_documents}$$

.

These are two competing goals that we want to improve at the same time. Given the need of the SMEs to find a very precise document, intuition says recall is more important, though if the searched document is not in the recalled documents, the user will try different search terms or give up. Similarly if the searched document is retrieved but buried under too many not relevant documents it is considered a low precision case.

1.3.3 Accuracy

Accuracy is the measurement of all correctly identified cases. It is used when True positives and True negatives are important. For our system, accuracy is how many relevant documents were returned from the DB.

$$\frac{returned\_relevant\_documents + returned\_irrelevant\_documents}{total\_documents}$$

.

1.3.4 F1-score

F1-Score is when an equal weight is given to precision and recall. It is used when we want a better metric of the incorrect cases and when the classes are imbalanced. Other F-scores consider an harmonic mean giving different weights to each metric.

$$2 * \frac{precision * recall}{precision + recall}$$

.

1.4 Personal motives

Having worked in the VRD world for more than a decade, I have frequently heard of the utmost importance of using ESQs in the search engine, and how difficult is to keep them up-to-date and relevant. Implementing an automatic discovery system for these facilitates the inclusion of new equivalent terms and drives the process for the experts, switching from reactive and on-demand to proactive and periodical. The broad research effort done at the start of this project to decide the best approach also serves as exploratory work to be used as part of a currently on-going effort to improve the overall search system.

1.5 Objectives

- General Objectives
  - Automatically detect and select ESQs to enhance query expansion.
- Specific Objectives

- Create word embeddings to automatically detect ESQs given user behavior data logs from the search engine.
- Use an alternative word embedding system for terms that weren't seen when creating the original embedding model, known as Out-of-Vocabulary (OOV).
- Find mechanisms to select the best ESQs without resorting exclusively to similarity between vectors.
- Evaluate the found ESQs.

1.6 Methodology

Despite the academic approach of the current project, it is strongly business oriented since it stems from an existing corporate environment that deals with information of the vehicle repair world.

I will use a standard methodology for the memory, explaining the motivation, providing with a summary of the state of the art and giving brief explanations on the main topics and techniques, finalizing with the actual proposal and conclusions at the end.

For the development of the project, there is a software project created and hosted in GIT version control repositories along the queries used to mine the data, which were taken exclusively from the production log servers and properly anonymized. Any personal information identifier *(PID)* was removed, and the identification ids of any document masked.

The CRISP-DM [2] open standard model was followed for the project creation. User stories and tasks were created following Agile standards following the Kanban [3] variation.

Interviews and feedback were conducted as necessary over the different iterations of the project with database specialists, Quality Assurance (QA) experts, automobile repair SMEs and product team experts. An oral presentation supported by slides will be performed in the company's office to explain the potential benefits and the findings after finalization. Following the agile methodology style, changes and adjustments were done in response to feedback and business needs without compromising the core goals and chosen methods.

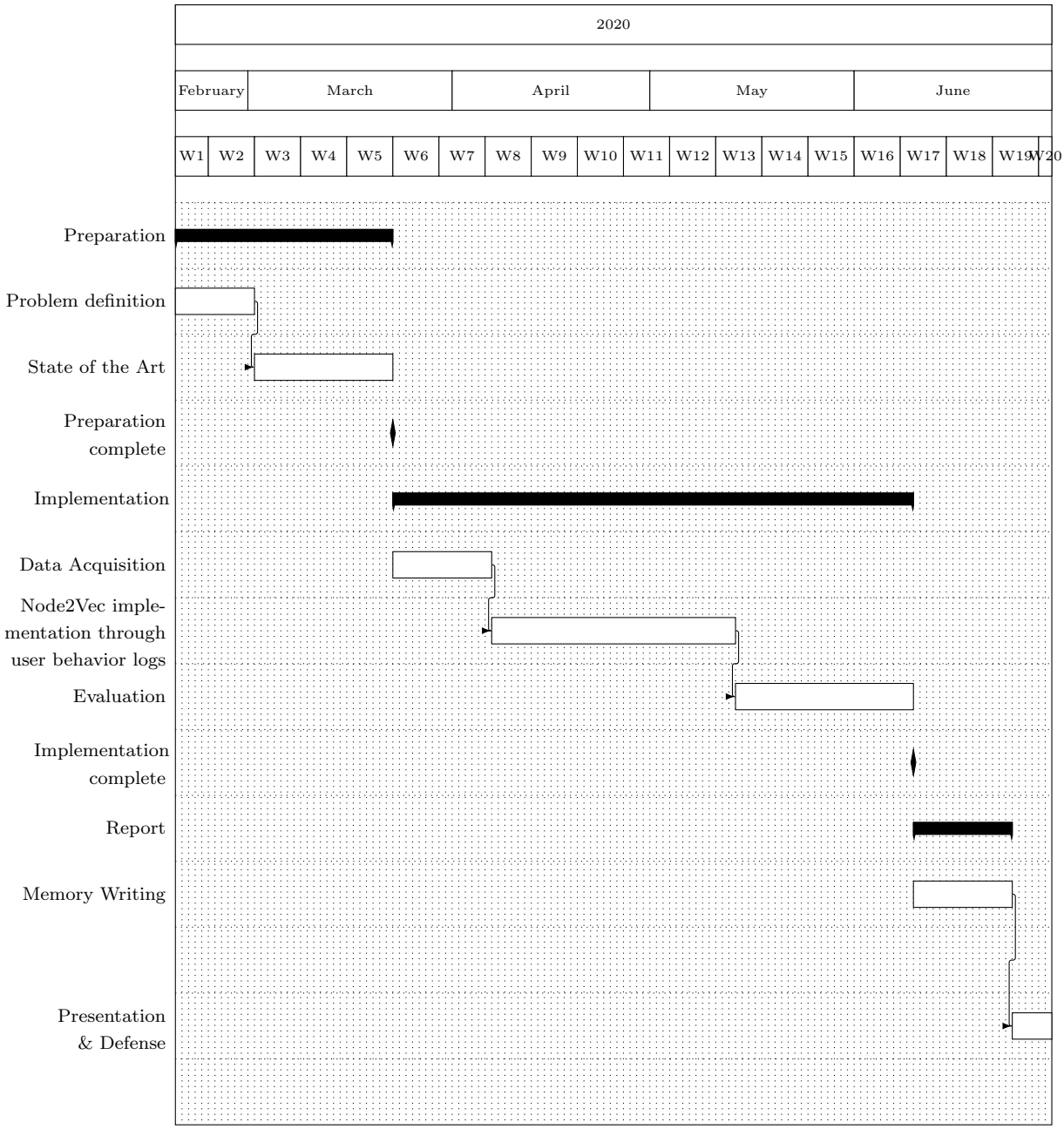
Jupyter notebooks were used to implement the different steps of the process, including data loading, preprocessing, cleaning and merging, model training, hyper-parameter search optimization, terms discovery by model, validation, metrics, and data visualization. Google drive was used to host all data input obtained from different sources, and the output generated

[2] https://www.ibm.com/support/knowledgecenter/SS3RA7_15.0.0/com.ibm.spss.crispdm.help/crisp_overview.htm CRISP-DM

[3] https://kanbanize.com/kanban-resources/getting-started/what-is-kanban/ Kanban

in the different parts of code. LateX was used to write the thesis' memory, using Overleaf [4], a cloud hosted LaTeX editor.

1.7 Project planification



[4] www.overleaf.com Overleaf

Chapter 2

State of the art

In this section a brief introduction to basic concepts and history of NLP will be given, followed by an explanation of synonym discovery and lexical semantics, to finish with an overall view of query expansion techniques.

2.1 Introduction

Language is not an exclusive notion to humans. Several animals have different forms of communication that are analogous to it. What makes human language special is a key feature discovered and largely studied by a generation of linguistic academics in the 50's that can be summarized in the Integration Hypothesis [1]. This theory states that human languages are composed of lexicon and expression, words and phrases with syntactic and semantic meaning, and that the added ability to merge and combine the building blocks turns this initial finite set into infinite potential combinations. This is an important characteristic that makes studying natural languages a significantly hard problem. Because of this, automatic methods to study and process them were developed in modern times, giving birth to NLP.

2.2 History of NLP

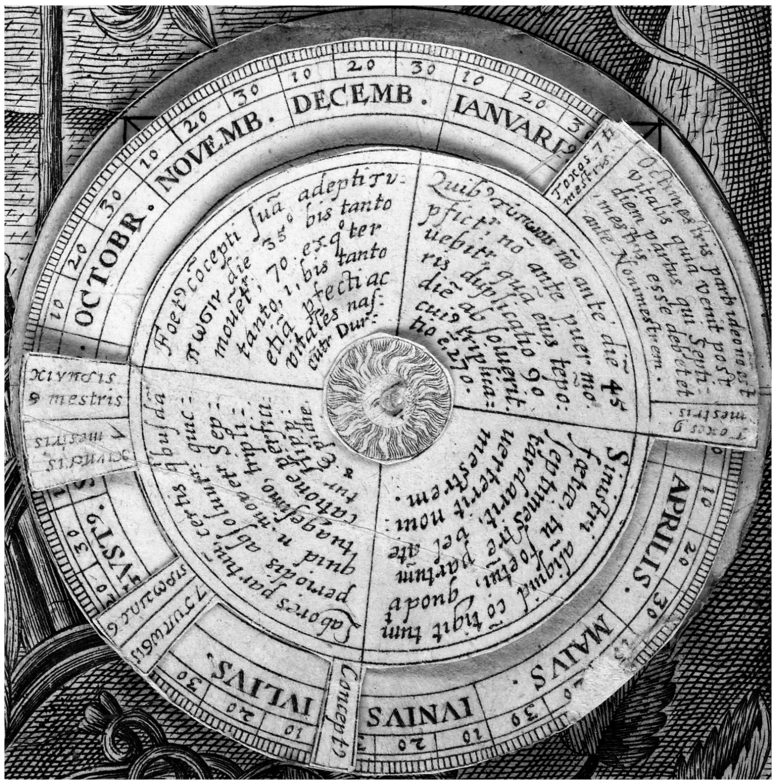
NLP is a combination of theoretical and practical knowledge. Part of it can be defined as the study of natural languages using automatic processes through computers, a field called Computational Linguistics. It is also about processing natural language, e.g.: automatic translation. It is considered a sub field of linguistics that binds all computer and engineering related fields.

It can be seen as a function that ingests large amount of data in the form of natural language and outputs both structured information or more natural language inferred from the input.

2.2.1 Proto NLP

The study of language is not a modern concept or interest. Most religious texts consider the truth hidden in words to have God-like creation powers. In Barcelona's 13th century, Kabbalah mystic Abraham Abulafia created texts formed of combinations of the Hebrew alphabet by following a set of strict rules he studied from the *Sefer Yetsirah*, a book that explains how *Yaveh* created "all that is formed and all that is spoken" by combining letters according to sacred formulas. He claimed that this effort granted him unexpected wisdom by crafting insightful sentences that would be hard to come by following normal reasoning. This is one of the first natural language generation by way of strict formalized rules that we have record of, forming a rudimentary method of NLP.

Majorcan mystic Ramon Llull took inspiration from Abufalia and devised a *volvelle*, a circular paper artifact with movable parts, that wrote simple concepts in increasingly small concentric circles. Llull thought that by combining all simple concepts then all aspects of the deity, or reality, would be eventually exposed.



Source: https://www.cambridge.org/core/books/reproduction/generation-reborn-and-reformed/03E27D7D1AC4769EC8C2EBE2C143FE21

Figure 2.1: Volvelle - Generation Reborn and Reformed (Part II) - Reproduction

In 1666 the famous mathematician Gottfried Wilhelm Leibniz intended to create a machine

that would make use of every human notion, and through combination and logical operators find the truth behind any reasoning [3], moving from Llull's theological approach to a philosophical one. He would abandon this idea as eventually considered the mechanization of language an immature approach. The concept of mechanising language study would be revisited later by modern NLP.

In 1913, Russian mathematician Andrey Andreyevich Markov deconstructed Alexander Pushkin's *Eugene Onegin* novel by listing the first 20k characters of the book in a single line without punctuation or spaces and then arranged them in 10-by-10 grids. Afterwards he counted the vowels, consonants, and frequency of each combination. He was testing a theory of probability that he had been developing, where upcoming events where affected by previous ones, called chained probabilities [4]. He intended to prove that in natural language a certain letter was more likely to show up in the text than others, adding a stochastic approach to the study of language and a mathematical structure to text.

Not highly regarded in his time, Markov's work inspired Claude Shannon's wildly influential "A Mathematical Theory of Communication" [5]. This work describes a method to measure the quantity of information in a message, and set the base for a theory of information widely used in the digital era. As Shannon improved on the underlying statistical model for language generation, where each following letter was assigned the probability observed in English texts, his models were able to automatically generate language that progressively resembled the real one more closely.

2.2.2 The 1950s

Modern NLP started in the 1950s with the works of the brilliant Alan Turing [6] leading to the Turing machine and the Turing test, a translator from a ciphered language to natural language and a test to determine if given a conversation a machine can be considered to have intelligence. These can be considered in turn NLP methods themselves.

Over the next few years there was an interest on obtaining funding from the USA government for automatic translation across different languages which lead to the George experiment [7], a joint effort between the Georgetown university and IBM, using the newly created IBM mainframe to translate from Russian on the topic of organic chemistry. It used six grammar rules executed in order that involved direct translation, rearranging words and multiple ternary choices and a lexicon of 250 words. It was considered a successful project, leading to the the ALPAC (Automatic Language Processing Advisory Committee) [8] report in 1966, a committee of seven scientists that recommended a series of topics that needed to be supported for automatic translation to be successful. At the end of the project they were very skeptic of the feasibility of the task at the given time. As a consequence, funding was greatly reduced. This

came to be the most known event in history for machine translation, and set the pace for the decades to come.

2.2.3 First chatterboxes and ontologies - 1960s & 1970s

A few successful projects were developed in the 1960, notably SHRDLU [9], and ELIZA [10]. SHRDLU is named after the most common letters used in a Linotype machine. Words would flush and print the nonsense phrase ETAOIN SHRDLU. The system allowed basic interaction of geometric objects in a dimensional world by commands written in English. The system couldn't say which actions were possible or valid before trying them, and used a physics system for the in-world rules.

ELIZA was an artificial intelligence created at MIT by Joseph Weizenbaum and named after the fictional character of Eliza Doolitle in the written play of George Bernard Shaw's Pygmalion, given how the character is taught how to speak in a specific manner. It was built to prove the simplicity of interaction between human and computer. The system identified keywords and attempted to simulate a natural conversation by pattern matching and following different scripts, most notably Carl Rogers, a psychoanalyst who famously repeated back what the patients said. It was an important landmark on the history of NLP, being the first famous chatterbot and a system that could attempt the Turing test. It incensed the imagination and interest of many people who, including its creator, claimed to have become emotionally attached to the system frequently forgetting that it was an artificial system and not a real human [11]. This was called henceforth the ELIZA effect, where people would assign human attributes and behavior to artificial systems, a process called anthropomorphisation. This concept was not new, it was commonly used in old Greek mythology where poets, in their intent of making the Olympian myths more thrilling, would often assign human vices to the literary Gods. ELIZA has been referenced in several media of the pop culture since then.

In 1971 the Defense Advanced Research Projects Agency *(DARPA)* created the project Robust Automatic Transcription of Speech *(RATS)*, intended to detect if "a signal includes speech or just background noise or music" and when positive identify the language, if it is an individual or a group and "spot specific words or phrases from a list of terms of interest".

This was one of the first high level practical NLP works, used mainly for espionage and intelligence, and started the field work for voice recognition.

2.2.3.1 Early ontology work

Conceptual ontologies, also called conceptual parsers, were created during the 1970s, trying to bring logical sense and structure from the real world into computer data.

MARGIE [12] was created as a set of three different programs that try to understand language. An analysis program that maps natural language sentences into conceptual structures, feeding a memory program that would draw inference from these conceptual structures and a generator that would code the conceptual structures back into natural language. The work is based on the Conceptual Dependency theory , a model that attempts to make the meaning independent of the words by paraphrasing and drawing logical inferences. The motivation behind the theory was to devise a method that would stand up to the tests of the psychologists of its time and to generate an input that could be used by computers for automation purposes [13]. In this model, two different phrases with the same meaning would have a single computational representation.

The model tried to simplify language down to representational tokes. It uses four basic types:

1. Real world objects with its' attributes
2. Real world actions with its' attributes
3. Times
4. Locations

2.2.3.2 Chatterbots

Several more complex and popular chatterbox were also created during this time. One case of interest was the PARRY system, created by Kenneth Colby to emulate the behavior of a paranoiac. It was the first system to pass the Turing tests, and it was connected to ELIZA using the Advanced Research Projects Agency Network *(Arpanet)*, the predecessor of internet, and held a nonsensical conversation that proved the superiority of PARRY which is believed comes from programmatic complexity [14]. It would take decades for a new breakthrough to render both systems obsolete.

These among other chatterbot like Jabberwacky would participate in the Loebner prize, a competition for AI to try to behave as human as possible. This competition is controversial and receives strong criticism from recognized experts in the field [15].

2.2.4 Machine learning and supervised methods - 1980s & 1990s

Entering the 1980s, the superior computing power following Moore's law and the introduction of alternatives to Chomsky theories gave ground for a revolution that would shape the face of NLP to this day. The introduction of machine learning shook the ground of language processing.

The underlying idea that, instead of capturing the explicit rules of the domain, a mathematical system could be designed to learn by imitation and self adaptation created a rich field of possibilities.

Early models like decision trees were more similar to the if-else logical paths from the old semantic rules systems. The introduction of Markov models and stochastic probabilistic models produced more accurate results and were more robust when unfamiliar input were feed into them.

2.2.5 Neural Networks and unsupervised methods - XXI Century

Entering the modern times research has leaned on unsupervised and semi-supervised algorithms, which often return less reliable and harder to interpret results, but is abundantly compensated given the overwhelming amount of unlabeled data compared to the labeled one.

During the currently ending decade certain algorithms have become more popular due to their analytical capabilities and reliability with deep Neural Networks (NN) being the de-facto standard for language modeling [16] and parsing [17] among others. This is considered a new paradigm in translation, since it attempts sequence to sequence transformations without any intermediate steps, a broadly used approach in all fields of NLP.

2.2.5.1 Self-supervised learning

An approach called self-supervised learning where instead of using human labeled data the system uses naturally occurring available information is also used. A method that we will rely heavily in this project is using the context and embedded metadata as supervisory signals of the question "Is word A likely to show up next to word B?" [20].

2.3 Synonyms and term equivalency

NLP is a vast subject with several fields and tasks, ranging from lower level tasks such as word segmentation and stemming to higher level tasks such as semantic inferences. Even while this project deals with term equivalency in searches which is different to synonyms, we will focus on synonyms discovery and a few auxiliary related tasks that have an extensive literature and a large range of developed techniques to understand how we decided to detect term equivalency on searches.

The Oxford's definition of synonym is "a word or phrase that means exactly or nearly the same as another word or phrase in the same language, for example *shut* is a synonym of *close*". We will assume that n-grams can be considered a short phrase for our purposes.

Even while this formal definition leaves out context, it can be implied since natural languages rely on word order and context for meaning. According to the contrast principle any difference in form in a language marks a difference in meaning, therefore two different words never have the exact same meaning [2]. Common elements such as air and water might be referred as H_2 and H_2O respectively in a scientific context, but the similarity is enough for the reader to infer the same meaning from the text.

When a term can be replaced with another without changing any of the truth conditions of the phrase, they share the same *propositional meaning*. For certain linguistic work a more strict approach to synonym detection might be necessary. ESQs are more lenient in similarity restrictions since it is usually better to have higher recall than precision. This is due to users preferring to have the document in the returned results even while having to look for it through irrelevant results than having to think on a different SQs after not finding it. This assumption is reinforced by the project studied system's order of return and categorization system 1.2.2.4.

For the current project, being an auxiliary system of a search engine, we are interested in queries that can recall the relevant documents associated to the search. Most of these queries will fall under the same semantic field, a set of related words belonging to the same domain.

This means that we want to detect ESQs in the same semantic field, but use a broader scope of what a synonym might be, such as acronym expansion (DTC - Diagnostic Troubleshooting Codes), proper nouns (Daimler - Chrysler), codes to parts (B001 -¿ front left door sensor) or whole large n-gram substitution. In chapter 5 the actual types of equivalencies found with the chosen methods is discussed.

Even when the official repair documents have a more formalized language the previously mentioned hot-line archives are also indexed which generates informal mechanical world lingo. Detecting this type of synonyms is also relevant.

Similar words and related words are close concepts to synonyms. An example is tire and wheel. In certain contexts those words might work as synonyms or similar words, but the relationship is closer to related concepts. Given the characteristics of the domain we want to clearly distinguish between a similar word to a semantically identical word, since a large amount of car parts are going to be similar to each other, but it is highly unlikely that the mechanic is interested in anything else than the exact document that person might be looking for. Related words might be useful in the sense that it might lead to related documents, which might be a secondary way of finding the relevant document, since most vehicle repair pages have hyper links to other needed or related pages. When constructing the models these ideas will be taken into consideration.

Synonym discovery has two phases, candidate detection and candidate selection. In candidate detection, lists of possible word synonyms are generated, and most of the self-learning and

clustering method we will discuss further refer to this step. Candidate selection is commonly done through well established supervised categorization methods using already existing sets of synonyms.

2.3.1 Synonyms related research

Four different approaches for synonym discovery have been used for the candidate selection phase.

2.3.1.1 Historical User Data

Following a pattern of "search, can not find a relevant result, search again" this approach tries to capture the different queries a user would use to find a specific page. A similar approach would be seeing all queries used by different users to arrive to the same page. In the current project the latter is used. Using this gathered data, the approach implies using different machine learning techniques to either group the words together, predict a similarity score or answer the question "Is this word a synonym of this other word"?. Supervised, unsupervised and self-learning algorithms can be used for this purpose.

An example is [42] where the authors present an unsupervised learning algorithm to recognize synonyms using statistical data acquired through a web search engine. The method is a variation of PMI [2.4.2.2] and is bench-marked against Latent Semantic Analysis (LSA) methods [2.4.2.2].

2.3.1.2 Linguistic resources

There are several resources that offer lists of synonyms that can be used for comparison and testing purposes.

2.3.1.3 Lexical Synonyms

These are grammatical synonyms defined by the rules of the language, e.g.: using WordNet API [39] [1]. This approach is considered by some experts [2] a catch it all method, not accurate enough for business or academic purposes but a useful benchmark and data set generation tool.

An example is [43] where the authors propose an open library that given two terms six similarity and three relatedness metrics based on WordNet [2.3.1.3] in the form of numbers

[1] WordNet https://wordnet.princeton.edu/

[2] Stanford CS224N: NLP with Deep Learning — Winter 2019 — Lecture 1 – Introduction and Word Vectors, Professor Christopher Manning https://www.youtube.com/watch?v=8rXD5-xhemo

are returned. This can be used as a benchmark baseline for comparison purposes between the current work's algorithms and other existing literature.

Several sources of this type of data are available, some of the most important are described next.

⟨wordnet⟩ **WordNet®** is an online database of lexical resources hosted by the Princeton University. it works as "an online lexical reference system" inspired by current psycho-linguistic theories. It provides sets in the form of emphsynsets of synonyms and a discrete number of relationships among these sets called conceptual relationships, such as hyperonymy or meronymy.

It is widely used for computational linguistics and NLP. It is similar to a thesaurus, but it distinguishes itself by disambiguation of similar words and by labeling the semantic relationship among words.

Thesaurus.com [3] is the most well known source for synonyms and antonyms. Is widely used by poets, writers, researchers and linguists. Even while the results for a specific domain might not be the best for a business oriented app it is a valuable source of synonym sets for benchmark purposes.

2.3.1.4 Word embeddings

Described in depth in the following section due to its' importance in this project, this approach uses the natural written terms' nearest neighbours inside a context defined by a window to find how similar in meaning are the words to each other.

2.4 Lexical semantics

Lexical semantics is the linguistic study of word meaning. Abstracting the meaning of the word makes it possible to detect a similar enough meaning in one or more terms, leading to synonym, antonym (opposite meaning) or meronym (part-to-whole relationships) discovery.

2.4.1 Distributional semantics

A sub-field of lexical semantics is *distributional semantics*, based on the idea of the distributional hypothesis formulated by linguists in the 1950s [24] where synonyms are frequently surrounded by the same context and the differences between them is proportional to the differences in the context itself. This idea is the core idea behind the word embeddings technique.

[3] https://www.thesaurus.com/Thesaurus.com

The intuition of distributional semantics is that the meaning of a word is given by its usage, in the way of the immediate context. Philosopher Ludwig Wittgenstein, contemporary to the 50's linguists, wrote "the meaning of a word is its use in the language" [19].

2.4.1.1 Vector semantics

Based on Wittgenstein's linguistic and philosophical work, a methodology to represent this similarity in a hyper plane based on vectors was formed, called *vector semantics*.

By studying the context in word composed windows we can assign in matrices a similarity rating going from 1 for identical terms to 0 for totally orthogonal concepts. The closer a vector is to another in direction, the more similar the words they represent are, which means the cosine of the normalized vectors can be used as a similarity metric. The similarity is measured in the usage of the word, or it's meaning in the language. This distinction in contrast to lexical or other type of similarity is crucial.

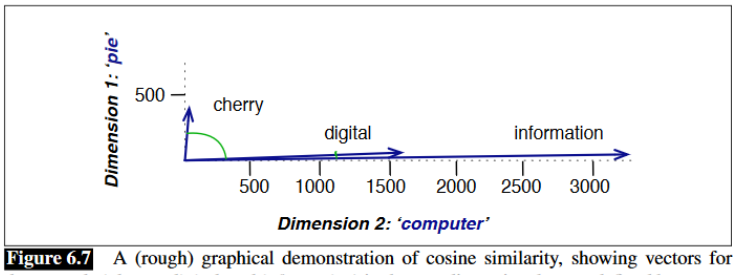


Figure 6.7 A (rough) graphical demonstration of cosine similarity, showing vectors for three words (*cherry*, *digital*, and *information*) in the two dimensional space defined by counts of the words *computer* and *pie* nearby. Note that the angle between *digital* and *information* is smaller than the angle between *cherry* and *information*. When two vectors are more similar, the cosine is larger but the angle is smaller; the cosine has its maximum (1) when the angle between two vectors is smallest (0°); the cosine of all other angles is less than 1.

Source: https:https://web.stanford.edu/ jurafsky/slp3/6.pdf

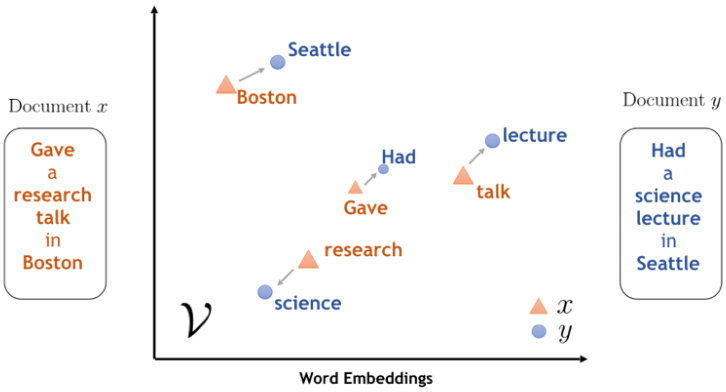
Figure 2.2: Vector Semantics. Source: Speech and Language Processing (3rd Edition)

Vector semantics are not only used for similarity purposes. Vector operations can be done to infer relational meaning. If the sum of the vector "Spain" and "Capital" point to "Madrid" in the hyper-plane of words, it has been observed that the sum of "France" and "Capital" usually falls very close to "Paris". In the same way, the vectors "Paris" - "France" usually falls close to "Capital". This concept has very powerful implications for lexical semantics, though we won't be using it in this project.

The most common methods to construct vector semantics is through word embeddings.

2.4.2 Word embeddings

Word embeddings is a type of self-learning language modeling technique used in NLP that maps terms to vectors. Each term from a prefixed size vocabulary is represented as a vector with real number coefficients in a predefined geometrical space using the text as input. This distributed representation is learned by usage of words in such a way that terms with similar meaning have a similar mathematical representation. Each term is considered to be *embedded* in the natural language text it occurs by being surrounded by other terms, and respectively embedded in the vector space near other terms with similar meaning.



Source: https://www.ibm.com/blogs/research/2018/11/word-movers-embedding/

Figure 2.3: Word Mover's Embedding: Universal Text Embedding from Word2Vec

Once the word embedding is learnt the similarity between terms and the threshold for synonym consideration can be done through the cosine as mentioned above. In contrast, in the bag of words model different words always have different representations regardless of their use in the input. Compared to other sparse, high dimensional representations such as one hot-encoding [2.4.2.5] the processing cost is lower and the generalization power is greater. Being that word embeddings are usually learnt through NN, the method is grouped with deep learning techniques.

It is considered as one of the key NLP accomplishments and has enabled the impressive performance on some tasks that rely on deep learning.

2.4.2.1 Word Embedding Techniques

The first hurdle any data scientist that wants to deal with NLP is that computers don't understand languages as we do, they only understand bytes. So we have to represent the words in a binary format that computers can understand and operate with. Word embeddings make

this transformation possible, since computers can understand and operate with numerical dimensions.

The main methods to generate word embeddings will be discussed in the following sections.

2.4.2.2 Sparse matrices - LSA

⟨LSA⟩ LSA is a theory and set of techniques to extract semantic information from a corpus of text and the terms that compose them by using the distributional hypothesis. The output artifact is a sparse matrix with the frequencies of each word created using Singular Value Decomposition *(SVD)*.

SVD is a mathematical operation used in linear algebra to factorize a matrix into three matrices. This decomposition has important and interesting algebraic properties and is widely used in several areas of science, engineering and data science. Its' details are outside the scope of this project.

LSA main methods are:

Bags of Words The simplest method available, this method was described earlier [1.2.1].

⟨tf-idf⟩ **Term-frequency / Inverted Document Frequency** This approach is a variation of the BoW method where a more complex measurement of word occurrence is calculated.

In Term-Frequency / Inverted Document Frequency (TF-IDF) the frequency where both related terms appear together is then divided by the inverted frequency across different documents. The intuition is that two words that appear together are similar, but if they are common words showing in most of the learning documents then this assumption loses strength. The more scarce and specific to a domain a word is and the higher frequency it appears next to a word the more similar they are.

Positive Point-Wise Mutual Information An alternate method called positive point-wise mutual information *(PPMI)* [21] is based on the idea that the higher the probability two words ⟨PPMI⟩ appear than we would expect when independent events are assumed, the more similar the terms are. In this method negative probabilities are then assigned a 0 for performance reasons, since those calculations do not enhance significantly the predictions [22].

Pros and Cons of LSA An important advantage of LSA is that a relatively small set of data can still generate good results. A secondary advantage is that the complexity is low, so processing times are short.

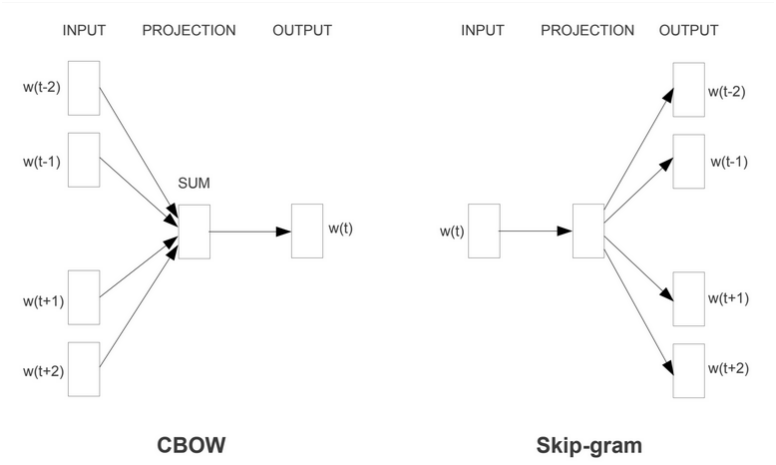
The main disadvantage is that any context information that would add semantic knowledge is lost. This makes this method a great fit for certain tasks and practically unusable for other. Sometimes N-grams are used to in this method to keep some important relationships between words. Another secondary disadvantage is that the created matrices are sparse, huge vectors mostly composed of zeros, which is a waste of space.

2.4.2.3 Dense matrices - Word2Vec

⟨word2vec⟩ For dense matrices we will study the *(word2Vec)* statistical methods created by Tomas Mikolov, et al. at Google [23] as an attempt to make the NN learning more efficient. It has become the de-facto standard for pre-trained word embedding and it was the starting point for vectorial extrapolation of concepts by way of spacial operations.

We can distinguish between the Continuous Bag-of-Words (CBOW) model and the Skip-gram with Negative Sampling (SGNS). These methods learns by using the context to predict ⟨skipgram⟩ the term. While CBOW uses a direct approach and learns the term from the context, SGNS learns by predicting the surrounding words given the distributed representation of the term and is based on the idea that a model can be trained to answer the binary question "Is word w likely to show up near apricot?".

This concept will be of assistance in the search of ESQs, since the higher the rating the more likely the words can be considered synonyms. According to the latest works [20] a large embedding window returns terms that are more related to the domain, while small windows tend to return terms that share the concept.



Source: "Efficient Estimation of Word Representations in Vector Space", 2013 [23]

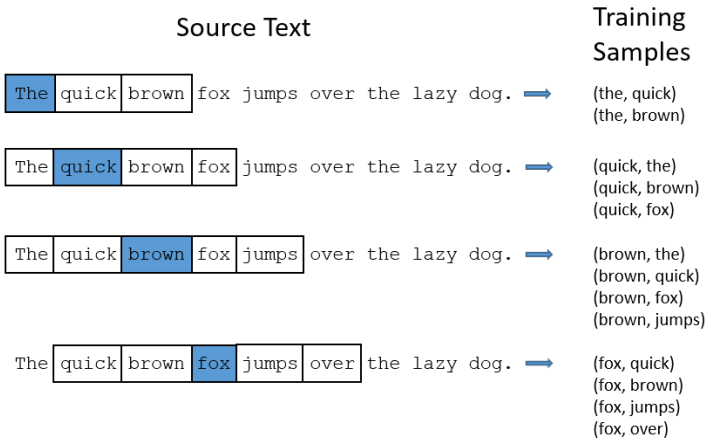
Figure 2.4: Word2Vec

Pros and Cons of Word2Vec The main advantage over BoW and TF-IDF is capturing the semantic meaning of words by context. A secondary advantage is that the resulting embedding vectors are much smaller.

CBOW Predicts the target word using the words around it, with a certain windows size.

Pros and Cons of Continuous Bag of Words The main benefit of the method is its' efficiency on processing time and storage which allows to process more documentation when available, usually returning a better output. The disadvantage is the added computing complexity.

SGNS Predicts the context of a target word using the words around it, with a certain windows size. It generates the possible pairs from the contextual words inside the window. Negative sampling refers to a tweaking proposed in the skip-gram authors second paper [44] where on each step only a small numbers of negative weights are updated when doing gradient scaling.



Source: "Word2Vec Tutorial - The Skip-Gram Model", 2016 [23]

Figure 2.5: Skip-gram

Pros and Cons The advantages are that by changing the window size the amount of data for training can exponentially increase with the same data, and also instead of learning directly from the co-occurrence matrix it compresses the components, leading to dense matrices with low dimension vectors.

A major disadvantage is that the resulting model would be very large with the initial proposition, the number of vector components multiplied by the vocabulary words. Therefore a number of tweaks have been proposed and implemented to scale down the size such as sampling the most frequent words or using negative sampling as mentioned above.

We won't delve in the problem of polysemous terms, where a single term can have multiple meanings, due to scope of the project and relative importance. For such a specific domain these cases can be easily recognized by the experts. It reminds a field of improvement for future work.

⟨glove⟩ **GloVe** *(GloVe)* is an an improvement over word2Vec developed by Pennington, et al. at Standford University[41]. It stands for global vectors and is an attempt to mix LSA methods and self-learning method like word2Vec. It is categorized as an unsupervised learning method.

2.4.2.4 Embedding layer

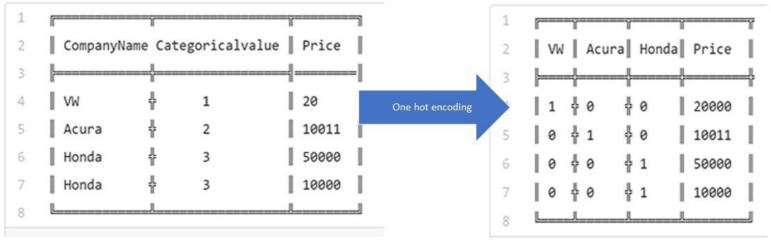
Embedding layer is a technique were the word embedding is jointly learnt along another NLP task that relies on NNs. The embedding layer is the front end of the NN model. The input are previously sanitized and prepared documents with methods such as one-hot encoding [2.4.2.5]. The vectors are randomly initialized with small numbers and incrementally adjusted as the NN method back-propagates. [40]

Pros and Cons The word embedding is learnt from the one-hot encoded terms. The downside of this method is that it requires a large volume of data and is expensive in processing time. The advantage is that it will learn the embedding for the data as a by product of the main task.

2.4.2.5 One-Hot encoding

⟨onehot⟩ This method, used in most word embeddings techniques, binarizes categorical data that was previously labeled in numerals. Normally, when dealing with categorical values a simple incremental integer is assigned to each value to pass to ML techniques that need to be fed numbers. The problem with this approach is that some of these models use Euclidean distance as measurement, meaning the category assigned to 1 would be worse or better than the category assigned to 2, category 1 + category 2 = 3, and the average between category 1 and category 3 would be category 2, $(1 + 3)/2 = 2$. All of these algebraic assumptions are fundamentally wrong and would wreak havoc in any prediction or interpretation of the model. The solution that one-hot encoding proposes is creating a multi dimensional matrix with the Cartesian product between all possibles values of all variables, and assign a 1 when the relationship exists and a 0 when it does not.

This solves the problem of categories equalling magnitudes and not having an Euclidean precedence between them.



Source: https://hackernoon.com/what-is-one-hot-encoding-why-and-when-do-you-have-to-use-it-e3c6186d008f

Figure 2.6: One-Hot encoding

2.5 Data sanitation and preprocessing

⟨sanitation⟩ Often the NLP algorithms have the need to reduce the amount of terms that will be used for training the models. Regular English has around 250K words, which will lead to a potential matrix of 62.5 billion entries. This is not considering the potentially infinite terms coming from *derivational morphology*, where the words are inflected to give contextual meaning to them.

A brief description will be provided of auxiliary tasks that are used on word embedding for preprocessing purposes. Since the underlying search engine does not use any of these, we won't delve deeply on them, but the usage on both the flagship product search engine and the current project, to simplify and improve recall and even precision through a higher ranking, will be encouraged to the product team after the project finalization.

2.5.1 Stemming

Stemming consists on reducing all inflected and derived words to the root form of the word. This is done to group a series of words into a single concept which is specially useful for categorizations where the exact meaning of each word is not important, e.g.: document categorization. It is usually performed as a preprocessing step to simplify the dictionary of words.

2.5.2 Lemmatization

The Lemma, also called citation form, is the root form of a given word. Usually the infinitive form in English, lemmatization is the process to group together said word with all of its' inflections. English in particular is a language with simpler and more streamlined inflection rules, but most Indo-European languages have many variations.

This process is closely related to stemming, the difference being that the earlier identifies the part of speech using the context, and has more success finding the meaning of the word in the document.

2.5.3 Morphological segmentation

Morphemes are variations of the same word, by way of inflections like conjugations which modify the verb according to context and the meaning, declensions which modify mostly nouns but potentially everything else like pronouns, adjectives, etc; and derivations which create new words by way of others, e.g.: compositions. [18]. This is strongly related to lemmatization and stemming as well.

This sub field of NLP identifies the classes for morphemes after separating the words into their individual morphemes. This task greatly varies in complexity by the language, in English different conjugations can usually be treated as different words for simplification purposes, while in others it would mean dealing with thousands of possible forms for some words.

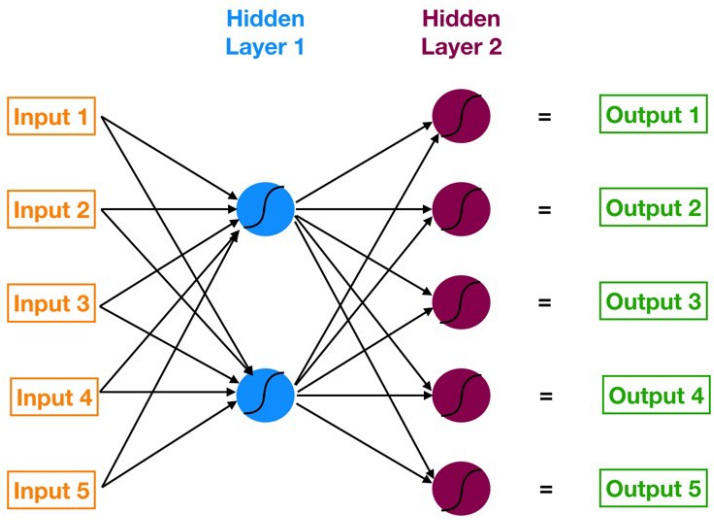
Stemming, lemmatization and morphological segmentation can be used to simplify the discovery of synonyms by considering all related morphemes to be the synonyms of the root terms.

2.6 Neural networks

As seen in both the history and classification of tasks neural networks have become the modern standard for NLP tasks given their modeling and predictive power.

Artificial Neural Networks (ANN) were developed were inspired by the analogous biological neural system in animal brains. As with other machine learning techniques, ANN are trained to perform certain tasks in a process called "learning" without explicit domain rules. Given enough cases of what the ANN is meant to learn, the system will recurrently adjust its own internal parameters to gradually diverge towards the best possible trained model that fits the given answers or the maximizing parameter that was set. This trained model is then used to predict new cases. This description somehow fits the category of supervised learning models but a new category called reinforcement learning was created from NNs. In reinforcement learning the models is adjusted to perform actions that minimize/maximize cost that are defined in a goal or series of actions. When an agent performs those actions a cost is given depending on the context. The model stochastically (randomly) decides to explore new actions or use known paths to minimize costs like in the simulated annealing heuristic. This behavior is architectured around a series of Markov's chains.

Their internal structure is a multi layer series of connected nodes called artificial neurons aggregated by said layers that conform a directed weighted graph. The multiple connections called edges across all nodes of each layer to the previous and the next one is in similarity of how real neurons transmit signals in the process called synapses. What is transmitted in this synaptic analogy instead of low voltage electric signals is data that serve as the input of several next level artificial neurons. Output is then calculated for each node as a non linear function



Source: Understanding Neural Networks
https://towardsdatascience.com/
understanding-neural-networks-19020b758230

Figure 2.7: Neural Network. Source: Understanding Neural Networks, Tony Yio

using the multiple inputs from the previous level which forms N-to-N relationships between nodes on adjacent levels. Each layer then performs its own type of transformation being a task more prone to empiric experimentation than theorizing given the complexity for a normal human to intuit what is happening mathematically inside the model.

Nodes and edges have weights that adjust the number being propagated increasing or decreasing the strength of each connection. Given that the starting points of the nodes of an ANN is data of any type including full documents and images the intuition is that as the model learns how to perform the task the connections gets stronger and weaker in complicated multi level intrinsical relationships that varies according to how strong the combined weights of the attributes get during the ANN level propagation. This ends up as a series of complicated relationships that follow the very important rule of smoothness, a small change in the input generates a proportionate small change in the propagated outputs, behaving like a gradient function. This allows backpropagation to slowly approach a better solutions as the model learns.

An important concept in ANN is hyper-parameters which are constants inside the model that set the set pace for learning rate, intermediate layers and batch size [46] that are usually calculated with other machine learning techniques.

The main interest in understanding NNs for this project lays in the fact that the coefficient weights for the vectors corresponding to each term are in fact the weights of the hidden layer of a single hidden layer convolutional NN generated when training that term. This brilliant

alternative use of a NN training inner implementation, used in other tasks such as image encoding, is what allowed to map meaning in words to a vectorial space.

2.6.1 Brief history of Neural Networks

NN started in the 1940s. Originally conceived theoretically [25] soon after a learning hypothesis around neural plasticity emerged [26] that was quickly implemented in 1954 [27]. With the creation of the perceptron the first implemented multi-layer network appeared in 1965 [28]. Continuous back-propagation [29] was derived from control theory [30] and dynamic programming [31]

In 1969 [32] made a discovery that would stagnate research for decades to come. Perceptrons can't process exclusive or (XOR) circuits which are used as the basic units of logic circuits and the computation power back then wasn't enough for useful projects. Still during that time some methods were refined and pragmatically implemented [33, 34].

The appearance in the 1980s of complementary metal oxide semiconductors (CMOS) brought the needed power for practical NLP [35]. But it wasn't until the beginning of the new millennium when graphical process units *(GPU)* and distributed computing brought the ability of performing thousands of parallel floating-point operations that innovations like using restricted Boltzmann machines [37] and higher levels of abstraction were possible [36]. By 2010, ANN were outperforming humans in several high level cognitive tasks [38]

2.7 Query Expansion

In this section we will explore past work done around QE. The technique is used mostly in standard IR but can also be used for cross-language IR, information filtering, plagiarism detection, etc.

The main motivation around query expansion is to improve recall in IR. There are several reasons why the document the user intends to retrieve might not be included in the recalled documents or lowly ranked in the return order. Assuming the document exists and is indexed, the reasons usually fall into:

1. Vague queries related to several topics, leading to poor precision.

2. Too short terms. It was determined that the average query is 2.4 words long [47].

3. The user might not be familiar enough with the domain of the document he intends to find, using the wrong terms.

QE is not limited to the case used by the current search system this project is based on, where all ESQs are appended to the FTS query with a logical OR. This case is used to plainly improve recall, but a logical AND can be used to improve precision when dealing with a scoring system or for query disambiguation, both can be used at the same time as long as the scoring system allows it to improve both metrics, the terms can be used for complex eXtensible Markup Language (XML) queries, top terms in document considered, etc.

Queries by user intention usually fall into one of these categories [49]:

1. Informational: broad topics with a large number of results.

2. Navigational: when looking for a website.

3. Transactional: when the user has the intention to execute an activity.

For the purposes of our system, the queries are informational with the added difficulty that the user demands high precision.

We need to distinguish between manual QE, where the user itself just adds or reformulates his search, automatic QE, where the underlying system automatically adds terms by default or given certain circumstances, and interactive QE, where the system proposes alternate queries and the user iterates until the result is satisfactory. For our project, the system is a conditional automatic system where the ESQs are only added when the set of returned document by the original query is empty.

QE has several problems, the most notorious being its complexity and potential added processing time to a search engine. In some cases, the solutions might not be feasible given the resources, a main concern of any engineering organization, or the added response time in a user oriented service can render it unusable.

2.7.1 The Vocabulary Problem

⟨vocabulary⟩

Most search engines work using indexes or ontologies, and both systems operate by matching identical words which leads to the vocabulary problem. This problem is defined by a mismatching set of terms used for IR that prevent the right information to be recalled, generated or exacerbated by the existence of synonyms and polysemous terms. This is the main reason to apply query expansion.

Several techniques to deal with this problem have been proposed, described in table 2.1.

Most of these techniques simply expand the original query with added terms as was discussed earlier, but some of them remove of modify some of the original terms of the query.

Method	Description
Relevance feedback	Any technique were the user provides with extra information after the query was executed
Interactive Query Filtration	Similar to interactive QE, in this case the system also learns as the user provides feedback, making it a specific case of relevance feedback
Corpus Dependant Knowledge Models	Models trained with specific corpus to extract meaning. Finding synonyms with word embeddings on official repair vehicle documents falls in this category
Corpus Independent Knowledge Models	Related to linguistic models, usually a stochastic model is created to calculate the chance that certain event will happen for prediction purposes. In this case could be which word is more likely to appear next to other and add the extra terms. Strongly dependant on language.
Search Result Clustering	Utilizing regular clustering techniques to group search results, and analyzing either the terms contained in the documents or the search logs that lead to those grouped documents.
Word Sense Disambiguation	Finding the semantic difference in otherwise identical words. Knowing if by "mouse" we are referring to the animal or the accessory.

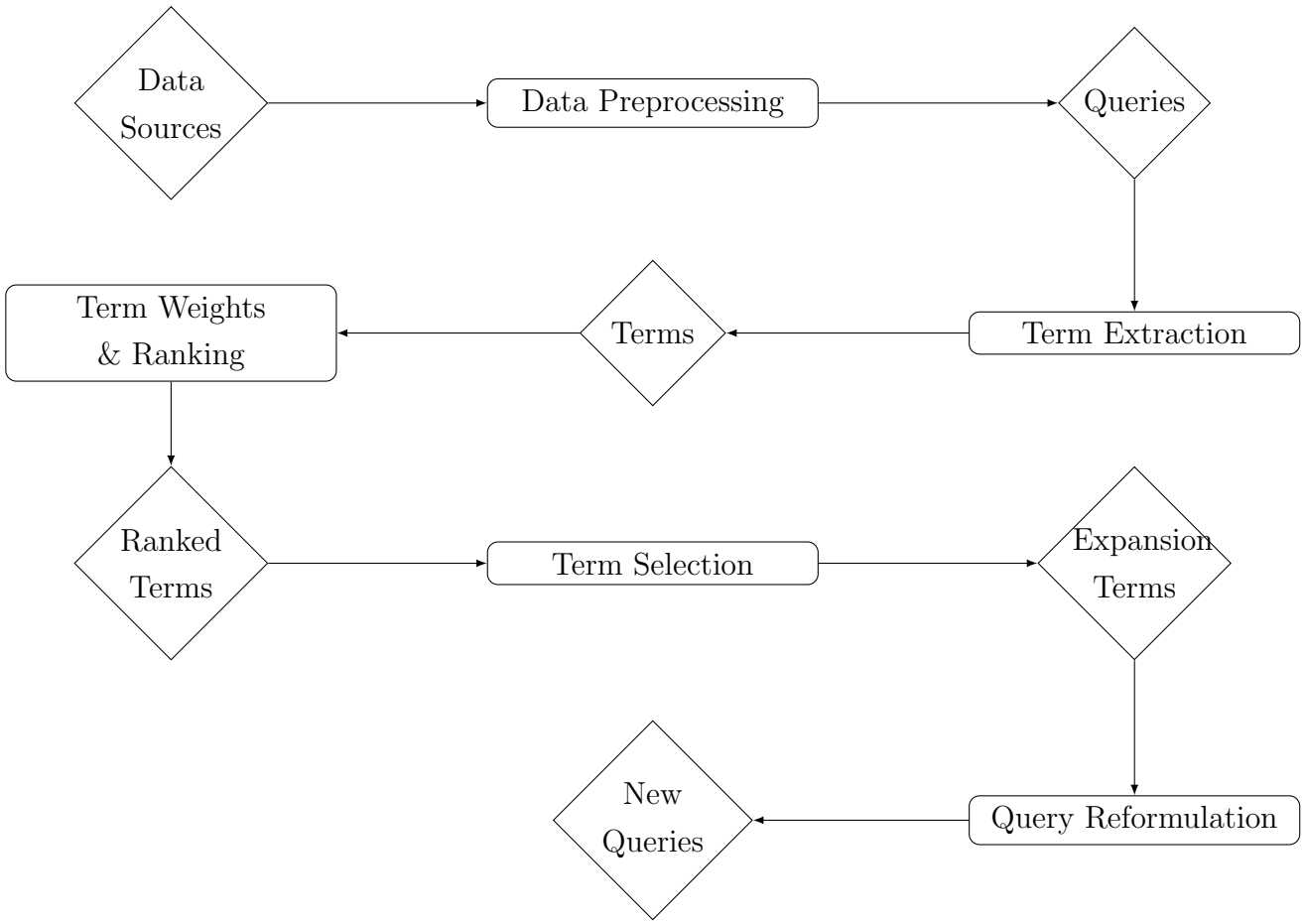
Table 2.1: Vocabulary problem methods.

⟨voc_methods⟩

2.7.2 Query Expansion Workflow

⟨qe workflow⟩ QE follows a certain series of steps. As with any other method workflow not all of those are mandatory or have the same complexity in different implementation but is works as a good guideline.

After getting the initial data and doing preprocessing work to get it ready, the terms extracted, ranked and selected according to the algorithm scoring criteria, the ESQs are used in the original query to reformulate it into a new one.



This model assumes for ML methods the case where the model for prediction or weighting haven't been trained yet. Otherwise, it is a direct arrow from query to predicted ESQs, using the trained model as the generator.

2.7.2.1 Data preprocessing

Usually consists of:

1. Text extraction from source
2. Tokenization

3. Stop word removal, stemming, trailing, etc.

The preprocessed data and the original query are then passed on the pipeline.

2.7.2.2 Term extraction

This step is often a sub product of the previous step, that generates a vocabulary of extracted terms.

2.7.2.3 Term Weights and Ranking

As discussed earlier when NLP or any ML technique is involved in this step, this is an asynchronous repeatable step that either trains for the first time or retrains the model used for prediction with the latest set of data. When applicable, hyper parameter tuning and model evaluation with a split of training and test data will also be done at this time.

In [52] the authors classify the techniques according the relationship between the term and the expansion:

- One-to-one, expansion term to query term.
- One-to-many.
- Feature distribution, where the top weighted terms of the original most relevant recalled documents are used for prediction.
- Query Language Modeling: Selects most likely terms based on a statistical model.

This step can be as simple as stemming the original query words or as complex as needed. Depending on the case a more complex solution is not necessarily a better one. It is also highly dependant on the search engine and the knowledge domain.

2.7.2.4 Term Selection

This seems to be a somehow controversial step. It can be seen in the literature [53] that this step was originally neglected, and over time its' importance became clear.

The different authors that have used a form of selection have no clear consensus on how many to return, going from a few [54] to a few hundreds [55].

In any case, given the large number of potential found terms, a further sub selection is sometimes a necessity, and always a good practice.

2.7.2.5 Query Reformulation

As with all other steps, there is extensive literature on different approaches for this. Some were mentioned earlier, like Boolean query, XML, structured query, and mathematical methods using the weights to replace each term.

For our project, this step is simple and straightforward. The candidates are all appended with a logical OR to improve recall.

2.7.3 Brief History of Query Expansion

The first work related to QE was done in 1960 by Moron and Kuhns [50] where the authors proposed a literature indexing technique for a mechanized search library, where a computer calculated a relevance number for a document based on terms. Referred as "Probabilistic Indexing" and based on "semantical closeness" by the authors' own words, this proved to be groundbreaking work relating the then young NLP work and search systems in an era when dealing with very large numbers of results in the millions wasn't an everyday problem.

Rocchio [56] proposed the first relevance feedback system. His work was further expanded by techniques such as term co-occurrence and cluster based information retrieval. All of this happened before the large WWW search engine era, so none of these techniques had to deal with the colossal data and all had significant success. In the modern web, those techniques have good recall rates, but abysmal precision. This called for a modernization of QE.

Manning et al. [57] details in his book a many deal of techniques for QE in the internet times, while Carpineto et al. [52] goes over the most important ones. Bhogal et al. [51] explains ontology based work. Since then extensive research has been done and a review on all this knowledge is overdue barring some PhD work [48].

Several state of the art enterprise search engines solutions provide with a built in QE system, like Lucene, MySql, Solr, Google Enterprise, etc.

2.7.3.1 Research and preparation

⟨research⟩ To finalize this chapter, we will mention that most of the query expansion techniques described in this chapter were considered at the start of this project. The main goal of the project as it is now is to enhance QE using embeddings via the company's user data logs taking advantage of the fact that it is a hard to come by commodity, but there is other valuable and accessible information that could have been used.

The table 2.2 details the most relevant alternate methods researched.

For those cases, data was acquired and working prototypes were coded and briefly analyzed. Early research on ontologies was done, which has a large corpus of literature in the topic [51].

Researched Method	Pros	Cons
Use official VRD for synonym discovery by word embeddings	Large available set of data	Doesn't capture user behavior
Group queries done by the same user in a short period of time, considering them different attempts to find a single document	Considers behavior from the users	Complex graph model for the embeddings
Grouping the search terms used by the different users with typical clustering techniques	Proven method	Hard to determine relevant features

Table 2.2: Alternative problem methods.

⟨alt_methods⟩

Summarizing, research was done and proof-of-concept *(PoC)* projects were created for several potential approaches before making a decision. This preliminary work is also valuable by itself in the context of the project.

For timing constraints, potential and ease of use, the scenario where the graph representing different SQ used to reach the same document was selected. In [48] the authors explore a large number of QE related literature and reach the conclusion that most related works are done using search logs, which seem to return better results for specific domain knowledge which is a good fit for this project.

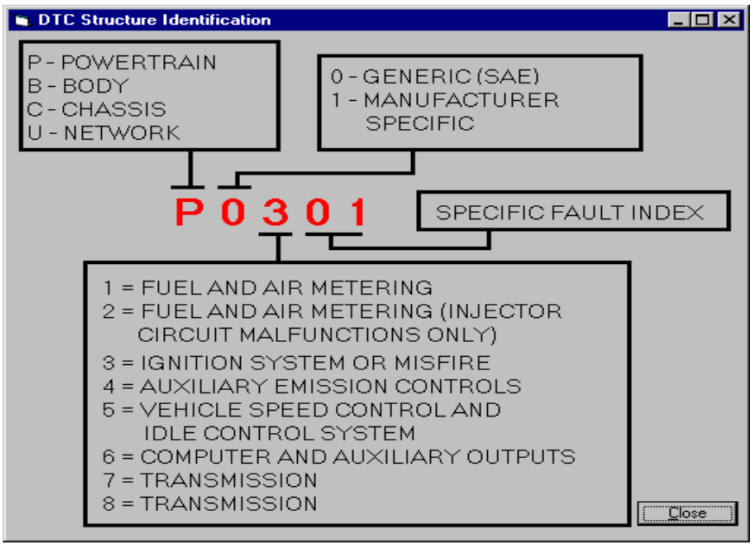
Chapter 3

Data Acquisition and Preprocessing

In this chapter I discuss how the data to train, test, validate and measure the model was obtained, along its format and schema. In the QE workflow chart [2.7.2], it corresponds to Data preprocessing and term selection.

3.1 Diagnostic Troubleshooting Codes

In earlier chapters the concept of Diagnostic Troubleshooting Codes (DTC) was mentioned but not explained. Before delving into this chapter we explain the concept of DTC, also known as On-Board Diagnostics Codes *(OBD-II)* [1] .
⟨obd⟩



Source: DTC Codes Tool, Solera inc.

Figure 3.1: DTC Codes

[1] url OBD Codes https://www.obd-codes.com/

In the automotive world, modern vehicle have very complex internal computers and electrical wiring that regulate several important aspects including sensors that help to troubleshoot potential problems. A scan tool is an external device that is connected to a vehicle and receives a series of so called P-codes that indicate specific problems. The format of these codes is formal and regulated, making it easy to identify them with regular expressions. Vehicle repair documents are full of references and tables of P-codes, and it is a very common occurrence that the codes themselves are used as search term. We did not include them in our data since we believe it will confuse the model, and they merit a separate research on their own.

3.2 Document queries

The most important set of input data for this project are the logs created in the system when a user performs a search with specific terms and select a document from the retrieved results.

Data was obtained using a combination of Splunk [2], a tool used to monitor large amounts of semi structured data for troubleshooting, metrics, alarms, etc, and relational databases using SQL. For simplicity's sake all persisted data sets are in the form of Comma Separated Values *(CSV)* spreadsheets.

3.2.1 Acquisition - Splunk logs

Splunk provides a front end website and an API that works with REST calls. Due to the size of the dataset, the latest was used, passing authentication to an endpoint with parameters.

This functionality queries large sets of data and tries to match fields and values given regular expressions on a non relational file database. These data schemas are known as semi-structured and are a good fit for logging and analysing very large amounts of data created from the daily operations. By querying the flagship product's log index [search query - chosen documents] tables were obtained. Multiple destination documents per SQ are joined by an underscore in the data.

The resulting data is exemplified in table 3.1.

3.2.2 Preprocessing

preprocessing⟩ A comprehensive process to clean and prepare the data to train the algorithm is applied to the initial data.

1. Remove any non alphanumerical character in SQs.

[2] https://www.splunk.com/en_us/about-us/why-splunk.htmlSplunk

Timestamp	UserId	Search Query	DocumentGuid
20200510000000	00001111	po625	
20200510000001	00001111	P0625	3a528506cc17
20200510000002	00001112	oil pan gasket	904dbd506029_6cf9e20226b0
20200510000003	00001113	fuel pressure	6cf9e20226b0

Table 3.1: Splunk Logs List [Search Query - Document Id]

⟨splunk_log⟩

2. Replace all characters with the lower case version when needed.
3. Remove all extra white spaces in SQs so in all cases a single space is joining different words.
4. Remove any trailing white spaces.
5. Replace SQs white spaces with an underscore, e.g.: fuel pump - fuel_pump.
6. Encode all strings to Unicode Transformation Format 8 *(UTF-8)*.
7. Split and explode the cases where more than one document is associated to a search term on a single row. Multiple rows will be created associating the SQ with each document.
8. Remove any row where the SQs are only composed of digits.
9. Split SQs that match the format of any number of P-codes [3.1].
10. Remove duplicate rows.
11. Filter out the documents with a single associated term.

Terms used to navigate to a more than one document are used in the graph model to find the similarity. Documents associated to a single term used can't be used for this purpose, so we eliminate them in the last step.

A quick summary of the initial data showed an average of 4.5 SQs associated per document, with some cases going as high as 200.

3.3 SMEs Created Synonyms

As we mentioned in the introduction, SMEs created synonyms is the list of synonyms that we intend to automatically maintain with this project. The company's databases contain this user created and curated list of ESQs defined loosely as synonyms and currently used for query expansion. The set is composed currently of around 300K pairs. This data will be used to

QueryA	QueryB
Engine	R/P Lock Valve
Engine	Radar CAN 1
Fuel Trim	Stop Cranking
Fuel Trim	Stops Cranking

Table 3.2: Subject Matter Experts' hand curated synonyms

⟨sme_synonyms⟩

match these existing ESQs with the ones found by the algorithm for scoring purposes during the model hyper-parameters optimization [4.1.2].

3.3.1 Acquisition - SQL Schema

The list of synonyms was obtained from a normalized relational SQL database. For simplicity's sake a mirrored data set was appended. This set is simply a copy where all rows of the form [Keyword,Synonym] swap the columns as [Synonym,Keyword], given the commutative nature of the ESQs.

The data set is exemplified in table 3.2.

3.3.2 Preprocessing

A set of preprocessing steps similar to the ones used in the logs data 3.2.2 was applied, excluding the steps where the document IDs are involved, such as exploding where multiple associations exists and filtering one-on-one term-document rows.

Once finished with the data acquisition and preprocessing, we have at the end of this pipeline step a set SQs composed of bound-by-underscore cleaned and filtered terms, and a clean list of SMEs' created ESQs.

Chapter 4

Term Weight, Ranking and Selection

4.1 Term Weighting & Ranking - Graph embeddings

In this chapter we will explain why we chose graph embeddings to find the similarity between SQs, how the model was trained and how the best possible parameters set for this was selected.

Embeddings are preprocessing methods that both reduce dimensionality and enable a computer to digest and operate data. Word embeddings are useful to detect similarity between the words' meaning by creating a vectorial space where each vector represents a word, and meaning can be inferred from the vector position. Once the vectors are created, the similarity can be calculated using the cosine between them.

While we can treat each SQ as a standalone sentence or append one after the other by grouping them by retrieved document, word embeddings are meant to be used in natural language text. SQs are not natural language sentences, certain key factors make them fundamentally different:

1. In natural language, very different sentences from a word-to-word standpoint can have very similar meanings. For search terms, different sentences can have very similar meaning but the retrieved documents be very different, since indexing does not consider meaning, only word presence in the index.

2. In search engine matching, the first word of a SQ is not closer or farther than the last one to the related document, since context and word order is not considered, e.g., "broken water pump" is exactly the same as "water broken pump" or "pump water broken".

3. Average SQ length is much shorter than an average natural language sentence length.

4. Articles, prepositions, conjunctions and pronouns are usually skipped.

Due to these differences, a graph representation, where each node is a SQ, was deemed a better fit. A simple graph can be modeled where the queries and documents are nodes and an edge between them represents that the document was retrieved with the SQ and selected. The distance from each of those queries is the same to their resulting documents, and the navigation from different documents using the queries as connecting bridges is analogous to the co-occurrence matrix.

Our intention is to find the relationship between SQs using the connecting documents that were selected. The closer to certain documents by search different queries are, the higher the similarity between them. This can be described as each SQ having a neighbourhood of its' own. This idea is a good fit for the application criteria of a specific implementation of graph embeddings called node2Vec [46].

4.1.1 Graph embeddings - Node2Vec

Node2Vec is an implementation of graph embeddings that creates said embeddings by training a deep learning model with the information in the nodes of a graph.

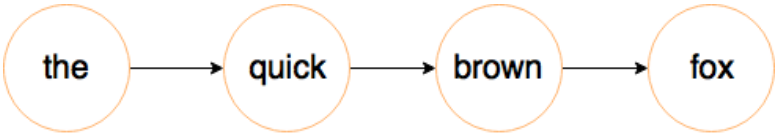
The graph created from the [SQ - document] tables are:

- *Undirected.* The [document-SQ] navigation is needed both ways to associate different SQ using the documents as bridges and vice-versa.
- Unweighted. The act of selecting a retrieved document is considered to have the same weight each time.
- Cyclic. This occurs naturally as many SQs can retrieve many documents and many documents can be retrieved from many SQs.

If the data to be embedded can be represented as an acyclic, directed graph, then the regular word2Vec algorithm 2.4.2.3, a well-known implementation of word embeddings, can be used since language sentences can be represented as such, where each word would be a node connected to the two words surrounding it, or to a single one if the word is the head or tail of the sentence. [This] - [is] - [a] - [graph].

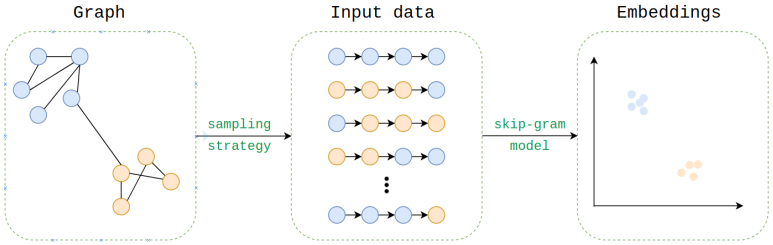
Node2Vec solves this by creating an embedding using word2Vec, while previously creating the needed directed acyclical graphs by way of random walks Random Walks (RW) sampling. Each node is selected and a random jump is done through an edge, while in the next step there is a chance to keep exploring new nodes or return to the previous node. For each node a number of random walks are created as each one navigates through the connected nodes.

Once the paths are created, the underlying word2Vec algorithm uses the skip-gram variant [2.4.2.3].



Source: "node2vec: Embeddings for Graph Data", 2018

Figure 4.1: Node2Vec sentence representation



Source: "node2vec: Embeddings for Graph Data", 2018

Figure 4.2: Node2Vec

Node2Vec also allows to associate different data types and conceptual nodes. The artificially and stochastically created sentences then can also be determined by edge weight along the parameters, making this technique very flexible and powerful.

4.1.2 Model hyper-parameters

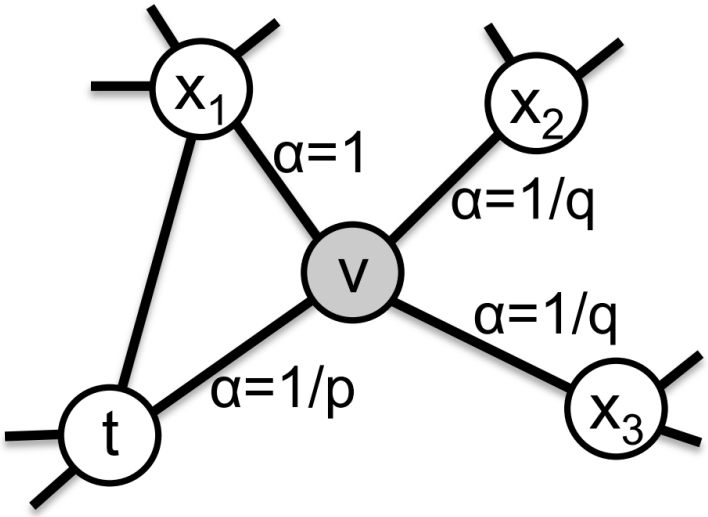
⟨hyperparam⟩ Node2Vec has four parameters that control how the random walks are created, which greatly affect the algorithm behavior.

1. N-components: How many random walks will be generated from each node.
2. Walk length: The length of each random walk.
3. Return weight: Referred as P, it controls the probability to return to the previous node when expanding a RW.
4. Neighbor weight: Referred as Q, it controls the probability to travel to a neighbor node different from the previously visited.

Between P and Q a preference for random walks to stay close to the node of origin or visiting farther nodes can be modeled. In the image 4.3 the RW has moved from node t to v, and it will randomly return to t or jump to a different node depending on P and Q.

This can be seen as using Breadth First Search for learning local neighbors and Depth First Search to navigate farther and learn globally, as shown in figure 4.4.

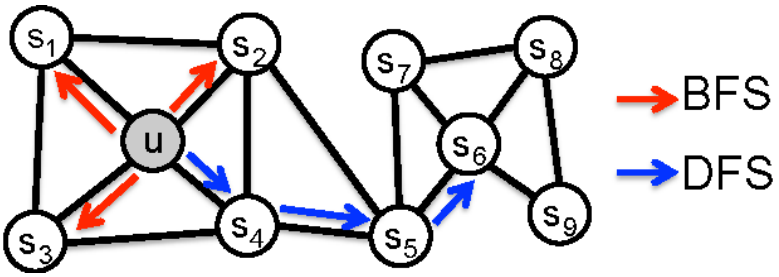
The algorithm also accepts the typical skip-gram parameters, like window size, negative score, and iterations. The default values were chosen to avoid excessive complexity.



Source: "node2vec: Embeddings for Graph Data", 2018

Figure 4.3: Node2Vec Parameters

⟨n2vec_param⟩



Source: "Graph Embedding for Deep Learning", 2019

Figure 4.4: Node2Vec DFS - BFS

⟨bfs-dfs⟩

4.1.2.1 Grid search

To find the best possible set of parameters a range of values for each parameter and a base estimator is necessary. Inside the base estimator, a scoring mechanism was coded to keep track of which set of parameters was deemed best.

The criterion is as follows: for each trained model with a different set of parameters, test if each ESQs found with a similarity threshold above 80% was present in the SME's synonym list. For each positive case, the score goes up by 1. The model with the highest score is then returned along the combination of parameters used to train it.

This allowed us to find the best set of parameters using an unsupervised algorithm, since there is usually no trained data to compare and score.

The ranges for the different parameters can be seen in table 4.1.

N components	Walk length	Return Weight	Neighbor Weight
32	2	0.25	0.25
64	3	0.5	0.5
128	5	1	1
-	10	2	2

Table 4.1: Node2Vec hyper-parameter ranges for optimization

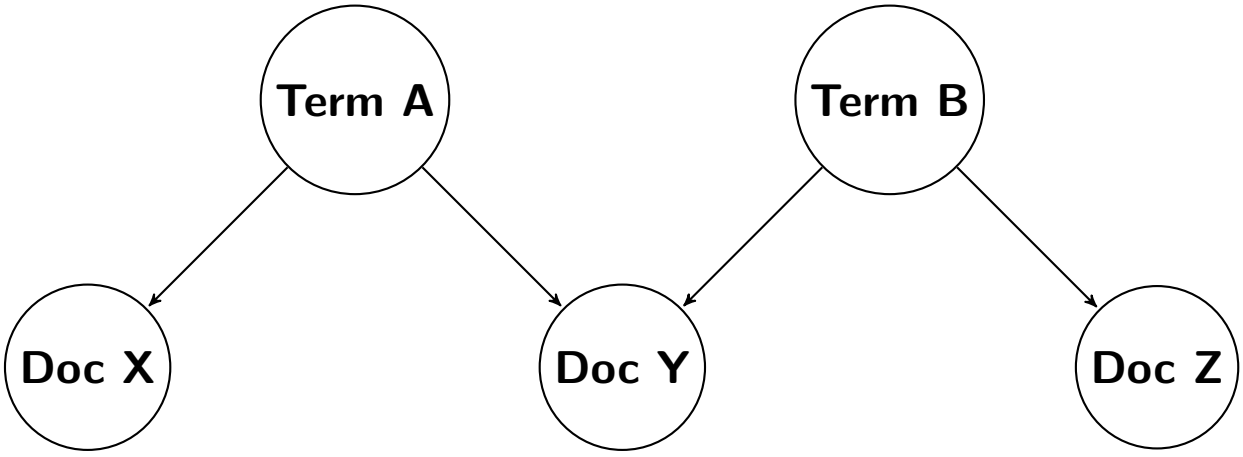
⟨hyper_ranges⟩

4.1.2.2 Parameter visualization

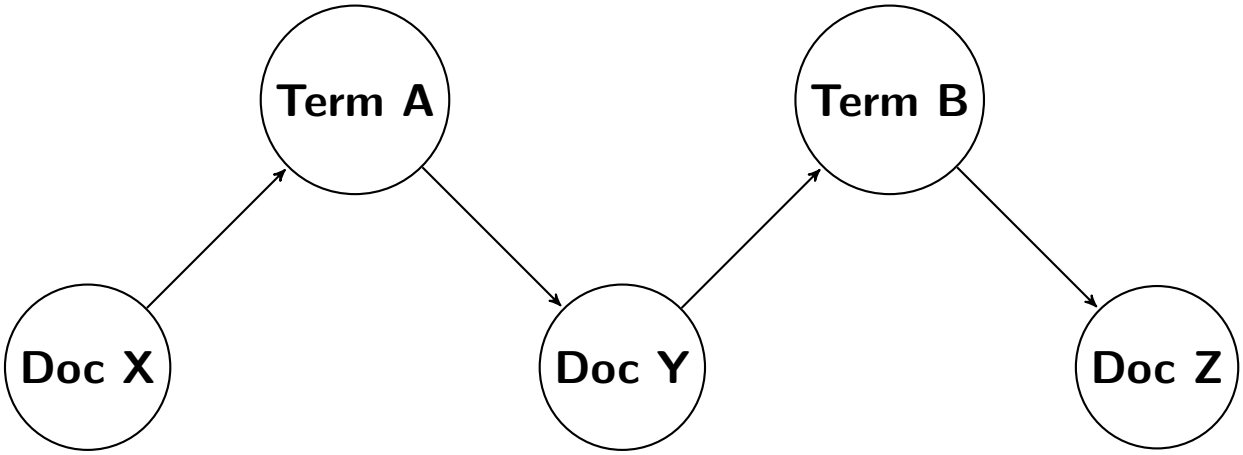
The set of parameters with the highest score was:

1. n_components: 64
2. walk_length: 5
3. return_weight: 1
4. neighbor_weight: 2

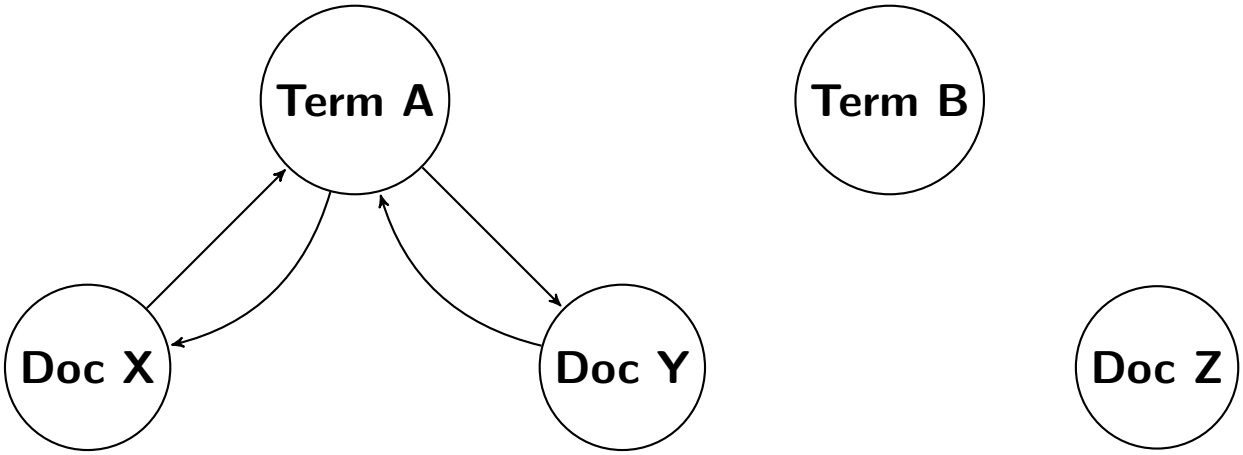
It is worth mentioning that the value of finding the right parameters is paramount for the algorithm functioning. This can be understood by visualizing how the random walks end up looking in the graph model.



This is how the data is originally stored, before building the graph. As we mentioned earlier the graph we end up building is undirected for navigation purposes, we need the documents to reach to their terms to find equivalences. Using a walk length of 5, the following case can occur:



This indicates a high chance of exploring different neighbors, since in every case the random walk chose to keep exploring, and will increase the similarity between term A and term B and between other terms related to documents X, Y and Z.



This indicates a high chance of returning to the same node, therefore localized results. In this case the term A won't increase the similarity with term B, but other random walks that relates doc X or Y with an hypothetical term C, will increase the similarity indirectly between term A and C, since there is a relationship between those parts.

The SQ and document nodes are equal from the standpoint of node2Vec, so RWs and vectors will be generated for each document. This information is ignored in this project but it can be used to associate documents between them for other purposes such as a recommendation system from one document to other, similar to online retail pages.

Hyper parameters visualization A compressed visualization of the score and the parameters' different values help to understand the impact of each one in the overall scoring

The following image is a screenshot of a 3D rendered plot that can be rotated inside an HTML page.

The size of the spheres corresponds to the number of components and the shade of blue to the length of the RWs.

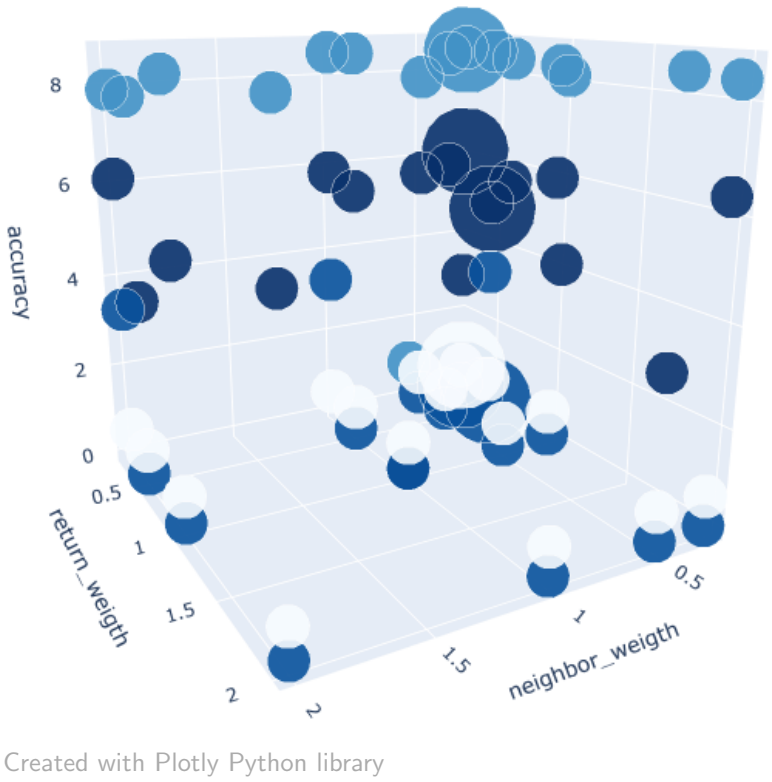
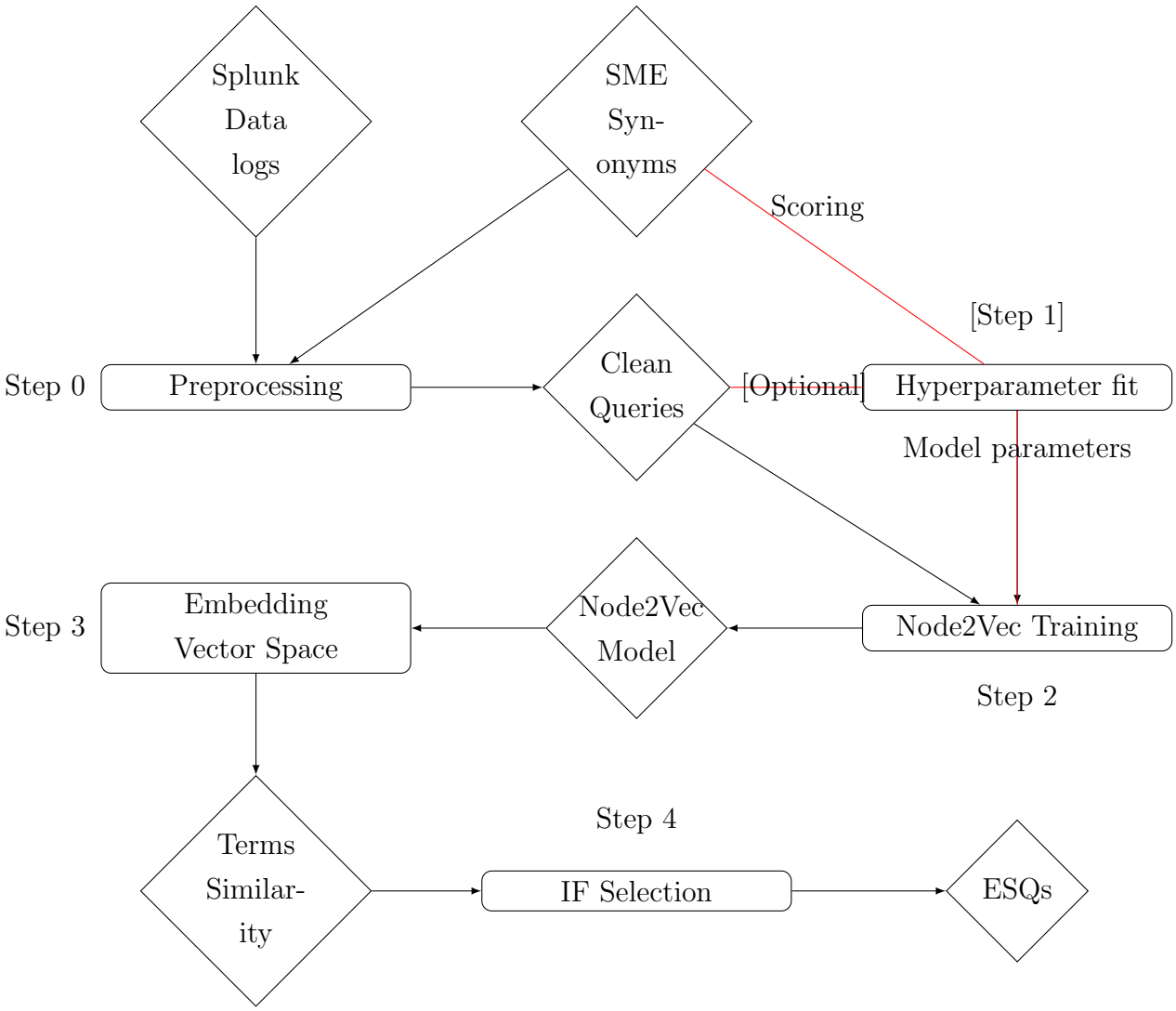


Figure 4.5: Hyper-parameter visualization

From the spacial study we can deduce that the number of components, P and Q don't seem to have a strong affect in the quality of the model, while the matrix shows very clear accuracy layers for walk length, with 5 being the best option and 2 and 3 degrading the quality significantly. It is interesting to note that having a value of walk length of 10 is much better than a low value but not as good as 5.

4.1.3 Model Training

The implementation steps follow a pipeline of Jupyter notebooks that exist independently of each other, where the different outputs work as the input of one or more of the next steps. The resulting workflow is a specific implementation of the one described in chapter 2 [2.7.2].



The graph used for node2Vec training containing the [SQs - Document] node relationships was created using the nxnetworks library, a Python library for manipulating complex structures [1].

The output of the workflow, a list of ESQs with a similarity score, was obtained by retrieving the 5 most similar queries for each query and adding them to the list of equivalencies only if the similarity was higher than a threshold of 90%.

4.1.3.1 Sentence Embeddings for Out-of-Vocabulary

An important problem to solve in word embeddings is how to process SQs that weren't seen before in the model. This was described as the vocabulary problem [2.7.1]. Since each SQ corresponds to a single vector in the space, OOV words do not have a vector and can't be

[1] https://networkx.github.io/ NxNetworks

compared.

Even while a solution would be to simply ignore those and retrain the model periodically with more data, a different solution was implemented. For OOV SQs a regular word2Vec embedding was trained and improved with each new term, using a stacked system that creates the embedding using byte pairs [45], a common form of data compression where the most common co-occurrence of characters are grouped, Glove [2.4.2.3] and character embeddings meant to detect misspellings. The Flair [2] library was used for this.

It is important to note that the nature of node2Vec and Flair stacked models are very different, and the ideal solution is to retrain the node2Vec model with the new SQs periodically. One noticeable difference from the OOV solution is that since it uses a regular word embedding technique plus a character embedding layer, a large number of ESQs are obvious misspellings and usually highly ranked in similarity, while in the node2Vec solution this is very rare, showing that the core idea of capturing the semantics of the terms and not syntax similarity was achieved. With this in mind, both models can be used to enhance the underlying search system, but with different focuses.

The results of the Flair stack embedded words weren't evaluated in this project. Its' only purpose is to provide a solution for all the processed terms.

4.1.3.2 OEM Specific Terms Discussion

An important problem that the mechanics face is that commonly each OEM refers to specific parts of the vehicle with a chosen term that differs from others. This was mentioned earlier in the introduction. Sometimes these differences are extended to different document repair types, by models, by years or by natural language evolution, where terms are replaced by others. One potential solution to find these specific cases would be to reuse the same node2Vec pipeline while reducing the window size of the skip-gram method progressively so specific synonyms can be discovered.

4.2 Term Selection

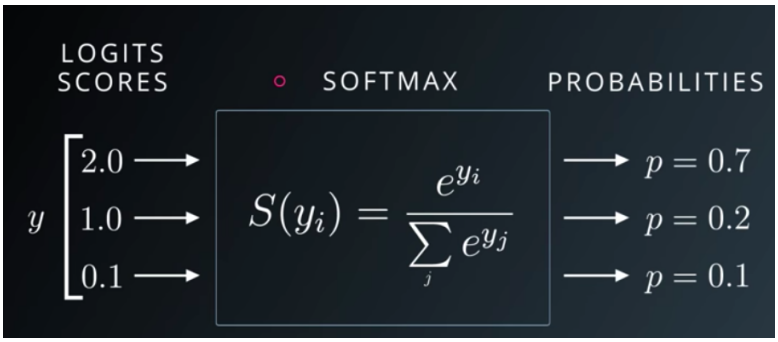
⟨selection⟩

As we will see in the next chapter, the ranking of the initial ESQs with highest similarity isn't very good, so a term selection mechanism was applied. Taking inspiration from TF-IDF [2.4.2.2], ESQs were split by terms, the frequency of appearance of each term from both sides of the similarity calculated, and a softmax function applied to the array of frequencies to smooth them out.

[2] Flair https://github.com/flairNLP/flair

4.2.1 Softmax

⟨softmax⟩ Softmax is a form of logistic regression that normalizes an array of values into a vector representing a probability distribution that sums up to 1, following an exponential curve. In our case we transform the SQs frequencies.



Source: "Udacity Deep Learning Slide on Softmax"

Figure 4.6: Softmax regression

The similarity rating was then divided by the softmax score, pushing terms with less frequent appearances to the top of the list and those with high frequency to the bottom without penalizing it too harshly. A new set of ESQs utilizing this term selection was tested.

$$ESQ\_Selection\_Score_{(queryA,queryB,similarity,queries)} =$$

$$\frac{similarity\_score}{avg(softmax(queryA\_freq, queries), softmax(queryB\_freq, queries))}$$

.

The ESQs ordered by selection score were deemed a better representation by the SMEs, mostly due to SQs with many relationships being pushed down in the selectivity. As an example, as an experiment further explained in the next chapter, data was obtained for a single vehicle model, the Chevrolet Silverado 1500LT, and the term "transmission" appeared in approximately 10% of the 7.000 found ESQs, and in the last 5% every single term included it. As a downside, terms that were specific codes referring to very specific parts were pushed to the top, e.g.: ['terminal 15' - 'hvac blend door actuators']. This types of codes are practically impossible to remember, but reliably finding the equivalencies for them at least for the most searched cases can be very valuable as the code works as an immutable centralizing point to all the different equivalent terms.

Chapter 5

Evaluation & Metrics

5.1 Evaluation

Evaluating search expansions is a complex task. One way of doing it is relying on SMEs' knowledge and ask for their opinion. Another would be to test if the found ESQs actually improve the recall. A set of tests, where given specific SQs certain documents are expected to be returned can be used to compare the expected documents with the ones retrieved by the equivalent terms. Unfortunately such tests do not exist at the time of writing so a small set of ESQs was handed to a group of SMEs whom categorized them and ran in the system the ones they were not familiarized with to compare the relevant results.

The list of terms to be evaluated was filtered to exclude:

- OBD codes, [3.1], actually split from the initial data.
- SQs that are similar enough that can be considered misspellings or morphological variations, e.g.: "transmission slipping" - "transmission slips". These can be useful as a simpler replacement of fuzzy matching techniques like using a dictionary of automotive terms with word distance or term reduction.
- ESQs where all the words of one are contained in the other, since these would return almost the same results. e.g.: "passenger seat belt tensioner" - "passenger belt tensioner".

5.1.1 Subject Matter Expert Equivalence Ranking

A reduced list of 200 highly similar ESQs was refined, split into four equal parts, its' parts mixed and merged. A different list of 100 of these terms was given to four SMEs for ranking according to a simple score criteria going from 1 to 5, 1 being non related SQs, 3 a situation where some related documents might be useful and some might not, and 5 an almost perfect

Measure	Rank [1-5]
count	194
mean	1.623711
std	1.131350

Table 5.1: Subject Matter Experts' Ranked ESQ Summary

⟨esqs_rank⟩

equivalence. The split and mix was done in such a way that each ESQ is ranked by exactly two SMEs.

This first list was generated from every possible vehicle in the system and all data types. No term selection was applied. This seems to have had an impact on the quality of the found ESQs.

The summary of the ranked ESQs can be seen in table 5.1

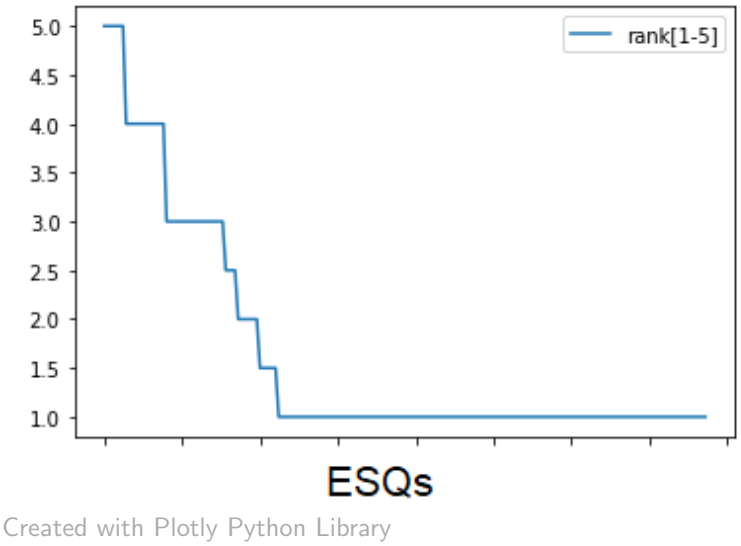


Figure 5.1: Equivalent Search Queries Matching Ranking

Looking at the data, we can see there is a large number of unrelated search queries, about 70% with a score of 1 and 80% of terms with less than 3. There is still potential for useful equivalences, but this suggests that the current system can be useful as an interactive recommendation system, where the SMEs that create the ESQs can iterate through the generated lists and decide which ones make sense. Future work might enhance the term selection system and achieve a higher degree of automation. Full unregulated autonomy seems like an ambitious goal that would need to be supported by other complementary ML techniques.

A more extensive overview of the found terms suggest three basic types and an important sub-type of ESQ:

- Semantically related sentences where the words are very different.

Semantic relationship	
blower motor output stage	hot air
ecu wiring	stalls while driving
wheel studs	torque specifications
Synonyms	
wipers always on	wipers stuck on
trans delayed shift	trans will not shift
Original Equipment Manufacturer Synonyms	
rear carrier fluid	rear differential clutch fluid
ac blend door actuator	ac temp actuator
Code to part	
dtc 2082	exhaust gas temperature sensor
4d32	oil level check

Table 5.2: Examples of found ESQ by type

⟨esq_types⟩

- Similar phrases where at most one or two terms are replaced, those terms being contextual synonyms.
- Similar phrases where the replaced synonym is a noun. This can be considered an important sub case of the previous one, given that is one of the motivations of the project, different named parts by different OEMs.
- Formal codes associated with the actual vehicle part it refers to.

An example of highly ranked ESQs can be seen to illustrate the mentioned types in table 5.2

5.1.2 Subject Matter Expert Feedback

The general consensus among SMEs was that some terms could have been ranked significantly higher in the context of a specific OEM or down to a smaller levels of granularity, like a specific vehicle. For this purpose, the pipeline was applied to to a subset of the data that only applies to a well-known car to the experts, a 2014 Chevrolet Silverado 1500LT. These terms were then further selected by dividing by the softmax-smoothed frequency [4.2].

The size of the input data wasn't enough to assure good results, and the resulting ESQs with were only a few, though the shown vocabulary was more familiar to the experts. Having access to the historical logs in a data lake can alleviate this problem, a current ongoing project in the company.

Following this idea, data can be split in any desired way to include any possible combination or years, makers, models, engines, transmission type, document types or any other category. It

www.Chevrolet.com, 2020

Figure 5.2: 2014 Chevrolet Silverado 1500LT 5.3L

is simple to do since it only implies changing the filters in the initial Splunk query, as long as the available data is enough for model training purposes.

Other important points that the SMEs mentioned are:

- Terms were generally difficult to rank due to being foreign terms or codes referring to specific parts that are hard or impossible to remember.
- Vehicles have become so complex over the years that a set of trained data for each OEM should be given only to an expert on that brand.
- Some terms belong to document types that the SMEs don't normally use, resulting in a lack of familiarity.
- Some terms were actually equivalent but in related areas different to vehicle repair, like electrical engineering or computers.
- The middle ranked ESQs were mostly parts of the vehicle. In this case, the relationship is valuable if the part has a prevalent problem in the model, i.e., it needs to be replaced often. Otherwise is too situational.
- Vague terms show up too much, and even while they have a relation with the detected equivalence, it is not useful for QE. This was mitigated with the softmax frequency scoring 4.2.1.
- The smaller the context the better the results. This is an act of balance with the amount of available data for the selection.

Search query	Document	Rank
terminal 15	37860717	1000
terminal 15	37893333	1000
terminal 15	37893482	1000
surge idle	37893482	900

Table 5.3: Search queries retrieved documents

⟨fts_terms⟩

After the ranking was done, the terms that the experts couldn't rank were searched along their related ESQs over the web, showing that in most cases there was an actual relationship. This shows that even while they are hard to validate by humans, the equivalent terms can be generally valid and be used in niche situations or for a different product, like creating a table of part codes related to a list of the actual part names used in the literature.

A conclusion that can be drawn from this is that while there might be a very large number of valid relationships, they are hard to validate and it might be overwhelming both for the experts and the search system. A practical business solution would be to focus only on the most searched terms for the most used vehicles, using a version of the Pareto principle where 20% of your cases yields 80% of the utility. This approach also indicates a more efficient working process if the expert would use the system interactively as mentioned earlier.

5.2 Metrics

A set of highly ranked ESQs was selected, and run through the SQL FTS database of the company to retrieve the sets of documents associated with each SQ. The data can be seen in table 5.3.

For each ESQs, the sets of documents associated to each SQ were compared, creating a confusion matrix where:

- True positive: a document found in the set of the original SQ and in the set of the equivalent SQ.
- False positive: a document *not* found in the set of the original SQ and the equivalent SQ.
- True negative: a document *not* found in the set of the original SQ *nor* in the set of the equivalent SQ. This set is the majority of the document space, about 37 millions. Tracking this number is not feasible in search system with large numbers of documents.
- False negative: a document found in the set of the original SQ and *not* in the set of the equivalent SQ

Given the large numbers detected when documents were returned, in the thousands or more, the confusion matrices were not useful. The metrics calculated at each step shown that the returned set of pages for both SQ of each ESQ was in most cases one of the following situations:

- Almost the exact same set of documents, regardless of criteria used.
- One SQ returned an empty set of documents and the other did not.
- The returned set were completely disjoint.

Due to this even while ignoring the false negatives the numbers for the confusion matrices were either almost 100% true positives, 100%true negatives or 100%false positives, so valuable information couldn't be inferred.

A second attempt was done using Mean Average Precision (MAP), a measuring technique that tells how many of the relevant documents are in the highest ranked retrieved documents. It was decided to filter by the FTS calculated rank, which is not used in the system at this moment. A series of different increasing restrictive criteria were used, experiment at each step, all the way to only getting the first five with highest word-count and highest ranking. None of the criteria proved useful, all were cases similar to those discussed in the confusion matrix.

A discussion with the SMEs was conducted to evaluate the possible reasons. The way the system evaluates return order 1.2.2.4 was signaled as a possible suspect. A future potential improvement can be getting the necessary document category and title metadata to group by category and order by words-in-title to compare each ESQ by these criteria using MAP.

In the cases where the ESQs returned an empty and a non-empty set, SMEs consensus was that in most cases the ESQs would be indeed useful since the non-empty set of documents was relevant to both SQs.

Chapter 6

Conclusions & Future Work

6.1 Conclusions

Looking at the evaluation it seems like the found ESQs have the potential to be usable but need manual intervention from the SMEs or an improved term selection step to become fully automated. If a periodically trained system is implemented that filters the input data with the most searched terms for the most looked upon vehicles the system can became valuable as it is, changing the current process where the SMEs passively create the ESQs following customer complains to a proactive one.

The pipeline can be streamlined with stored procedures and a master process that orchestrates the different steps to utilize different sets of logs of any size, including variations on the data for the graph. For each type of data a new set of optimal hyper-parameters can be found.

A persistent problem during the project was the limited cache of logging activity maintained by the current Splunk enterprise solution. While for immediate metrics and forensic purposes three months into the past is probably enough, for ML purposes it falls short. The limited numbers seems to be related to some of the low ranking in the terms' equivalence. This became obvious when retrieving data for a specific vehicle by adding extra descriptors. In those cases the solutions wouldn't find any ESQs with a similarity score higher than 70%. A solution where the logs are periodically poured into a data lake for future processing needs to be implemented for the success of this or any other data science project that rely on the logs.

As a final conclusion, while the overall numbers don't look very promising for a full automation, the general feedback was good, with a lingering feeling that both the current system has some downfalls that make this effort more complicated and that experimenting with some of the several ideas mentioned in this document could improve on the usability greatly, and potentially spin some side promising projects.

6.2 Future Work

Different sets of data can be used with similar pipelines, such as some of the mentioned in the state of the art chapter [2.7.3.1]. Having a better term selection mechanism would potential eliminate the need for human intervention.

Further expansion can be done by including different permutations of the original queries by swapping word order before binding them with the underscore. Extra preparation can be done in the form of word stemming, replacement by dictionary or byte-pair, though this would be more powerful if the same techniques with the same criteria are applied to the document BoWs passed to the engine.

ESQs would be more specific if being studied under the context of a single vehicle or different focused contexts driven by data splicing. The discovery can be limited to the most used terms and vehicles in the system, and data types and found SQs differentiated to be used for different purposes.

More parameter optimization can be done, specifically finding the best set for each splice of data, and changing the values for the underlying skip-gram model, specially on the windows size, to specify a search for equivalencies closer to related terms or to synonyms.

And finally, given that enhancing the search system is an ongoing important effort of the business, every NLP and ML technique researched and prototyped can be revisited for further development.

Bibliography

miyagawa:2014 [1] Shigeru Miyagawa, Shiro Ojima, Robert C. Berwick, and Kazuo Okanoya. The integration hypothesis of human language evolution and the nature of contemporary languages. *Frontiers in Psychology*, 5:564, 2014.

clark:1987 [2] Eve Clark. The principle of contrast: A constraint on language acquisition. *Mechanisms of Language Acquisition*, 01 1987.

liebniz:1989 [3] Leibniz G.W. *Dissertation on the Art of Combinations*. Springer, Dordrecht, 1989.

markov:2006 [4] A. A. Markov. An example of statistical investigation of the text eugene onegin concerning the connection of samples in chains. *Science in Context*, 19(4):591–600, 2006.

shannon:1948 [5] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

turing:1950 [6] A. M. Turing. I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, LIX(236):433–460, 10 1950.

dostert:1955 [7] W. John Hutchins, Leon Dostert, and Paul Garvin. The georgetown-i.b.m. experiment. In *In*, pages 124–135. John Wiley & Sons, 1955.

pierce:1966 [8] John R. Pierce and John B. Carroll. *Language and Machines: Computers in Translation and Linguistics*. National Academy of Sciences/National Research Council, USA, 1966.

winograd:1971 [9] Terry Winograd. Procedures as a representation for data in a computer program for understanding natural language. *AITR*, 1971.

izenbaum:1966 [10] Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1):36–45, January 1966.

izenbaum:1976 [11] Joseph Weizenbaum. *Computer Power and Human Reason: From Judgment to Calculation*. W. H. Freeman & Co., USA, 1976.

riesbeck:1975 [12] Neil M. Goldman Charles J. Rieger Christopher K. Riesbeck, Roger C. Schank. Inference and paraphrase by computer. *Journal of the ACM*, 22(3):309–328, 7 1975.

schank:1972 [13] Roger C. Schank. Conceptual dependency: A theory of natural language understanding. In *Conceptual dependency: A theory of natural language understanding*, 1972.

mauldin:1994 [14] Michael L. Mauldin. Chatterbots, tinymuds, and the turing test: Entering the loebner prize competition. In *AAAI*, 1994.

powers:1998 [15] David M. W. Powers. The total turing test and the loebner prize. In *New Methods in Language Processing and Computational Natural Language Learning*, 1998.

zefowicz:2018 [16] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *ArXiv*, abs/1602.02410, 2016.

choe:2016 [17] Do Kook Choe and Eugene Charniak. Parsing as language modeling. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2331–2336, Austin, Texas, nov 2016. Association for Computational Linguistics.

bubenik:1999 [18] V Bubekin. *LINCOM coursebooks in linguistics*. Lincom Europa, Munich, 1999.

anscombe:1953 [19] G. E. M. Anscombe. *Philosophical investigations*. Oxford : Basil Blackwell, 01 1953.

jurafsky:2019 [20] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (3rd Edition)*. Prentice-Hall, Inc., USA, 2019.

fano:1961 [21] R.M. Fano and Massachusetts Institute of Technology. Dept. of Electrical Engineering. *Transmission of Information: A Statistical Theory of Communications*. M.I.T. Press, 1961.

church:1989 [22] Kenneth Ward Church and Patrick Hanks. Word association norms, mutual information, and lexicography. In *27th Annual Meeting of the Association for Computational Linguistics*, pages 76–83, Vancouver, British Columbia, Canada, June 1989. Association for Computational Linguistics.

mikolov:2013 [23] Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

harris:1954 [24] Zellig S. Harris. Distributional structure. *¡i¿WORD¡/i¿*, 10(2-3):146–162, 1954.

cculloch:1943 [25] Pitts W. McCulloch WS. *A logical calculus of the ideas immanent in nervous activity*. Bull Math Biol, 1943.

hebb:1950 [26] D. O. Hebb. *The organization of behavior; a neuropsychological theory.* Wiley, 1950.

farley:1954 [27] B. Farley and W. Clark. *Simulation of self-organizing systems by digital computer.* in Transactions of the IRE Professional Group on Information Theory, vol. 4, no. 4, 1954.

akhnenko:1967 [28] A.G. Ivakhnenko and V.G. Lapa. *Cybernetics and forecasting techniques.* Modern analytic and computational methods in science and mathematics. American Elsevier Pub. Co., 1967.

dreyfus:1990 [29] Stuart E. Dreyfus. Artificial neural networks, back propagation, and the kelley-bryson gradient procedure. In *Artificial neural networks, back propagation, and the Kelley-Bryson gradient procedure*, 1990.

kelley:1960 [30] Henry J. Kelley. Gradient theory of optimal flight paths. In *Gradient Theory of Optimal Flight Paths*, 1960.

mizutani:2000 [31] Eiji Mizutani, Stuart E. Dreyfus, and Kenichi Nishio. On derivation of mlp backpropagation from the kelley-bryson optimal-control gradient formula and its application. *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, 2:167–172 vol.2, 2000.

minsky:1969 [32] Marvin Minsky and Seymour Papert. Perceptrons - an introduction to computational geometry. In *Perceptrons - an introduction to computational geometry*, 1969.

dreyfus:1973 [33] Stuart Dreyfus. The computational solution of optimal control problems with time lag. In *The computational solution of optimal control problems with time lag*, 1973.

nnainmaa:1970 [34] S. Linnainmaa. The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. Master's thesis, Univ. Helsinki, 1970.

mead:1989 [35] Carver A. Mead and Mohammed Ismail. Analog vlsi implementation of neural systems. In *The Kluwer International Series in Engineering and Computer Science*, 1989.

le:2011 [36] Quoc V. Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Gregory S. Corrado, Kai Chen, Jeffrey Dean, and Andrew Y. Ng. Building high-level features using large scale unsupervised learning. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8595–8598, 2011.

hinton:2012 [37] Geoffrey E. Hinton. A practical guide to training restricted boltzmann machines. In *Neural Networks: Tricks of the Trade*, 2012.

graves:2008 [38] Alex Graves and Jürgen Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In *NIPS*, 2008.

wordnet [39] Christiane Fellbaum. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.

goldberg:2017 [40] Y. Goldberg and G. Hirst. *Neural Network Methods in Natural Language Processing*. 2017.

nnington:2014 [41] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

turney:2001 [42] Peter D. Turney. Mining the web for synonyms: Pmi-ir versus lsa on toefl. In Luc De Raedt and Peter Flach, editors, *Machine Learning: ECML 2001*, pages 491–502, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

pedersen:2004 [43] Ted Pedersen, Siddharth Patwardhan, and Jason Michelizzi. Wordnet::similarity - measuring the relatedness of concepts. 04 2004.

mikolov2:2013 [44] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. 2013.

gage:1994 [45] Philip Gage. A new algorithm for data compression. *C Users J.*, 12(2):23–38, February 1994.

grover:2016 [46] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. *CoRR*, abs/1607.00653, 2016.

spink:2001 [47] Amanda Spink, Dietmar Wolfram, Major B. J. Jansen, and Tefko Saracevic. Searching the web: The public and their queries. *Journal of the American Society for Information Science and Technology*, 52(3):226–234, 2001.

iteshwar:2017 [48] Hiteshwar Kumar Azad and Akshay Deepak. Query expansion techniques for information retrieval: a survey. *CoRR*, abs/1708.00247, 2017.

kang:2003 [49] In-Ho Kang and GilChang Kim. Query type classification for web document retrieval. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, SIGIR '03, page 64–71, New York, NY, USA, 2003. Association for Computing Machinery.

maron:1960 [50] M. E. Maron and J. L. Kuhns. On relevance, probabilistic indexing and information retrieval. *J. ACM*, 7(3):216–244, July 1960.

bhogal:2007 [51] J. Bhogal, A. Macfarlane, and P. Smith. A review of ontology based query expansion. *Information Processing & Management*, 43(4):866 – 886, 2007.

arpineto:2012 [52] Claudio Carpineto and Giovanni Romano. A survey of automatic query expansion in information retrieval. *ACM Comput. Surv.*, 44(1), January 2012.

lavrenko:2006 [53] Victor Lavrenko and James Allan. Real-time query expansion in relevance models. *IR 473, University of Massachusetts Amherst*, 2006.

chang:2006 [54] Youjin Chang, Iadh Ounis, and Minkoo Kim. Query reformulation using automatically generated query concepts from a document space. *Information Processing & Management*, 42:453–468, 03 2006.

mitra:1998 [55] Mandar Mitra, Amit Singhal, and Chris Buckley. Improving automatic query expansion. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '98, page 206–214, New York, NY, USA, 1998. Association for Computing Machinery.

rocchio:1971 [56] Joseph Rocchio. Relevance feedback in information retrieval. *The Smart retrieval system-experiments in automatic document processing*, pages 313–323, 1971.

manning:2008 [57] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, USA, 2008.