Unified Processing of Natural Language and Relational Data

by

Andrei Stoica

A thesis submitted to the School of Graduate and Postdoctoral Studies in partial fulfillment of the requirements for the degree of

Masters of Science

in

Computer Science

Supervisors: Dr. Ken Pu, Dr. Kourosh Davoudi

Faculty of Science

University of Ontario Institute of Technology (Ontario Tech University)

Oshawa, Ontario, Canada

September 2022

THESIS EXAMINATION INFORMATION

Submited by: **Andrei Stoica**

Masters of Science in **Computer Science**

Unified Processing of Natural Language and Relational Data

An oral defense of this thesis took place on June 29, 2022 in front of the following examining committee:

Chair of Examining Committee	Dr. Patrick Hung
Research Supervisor	Dr. Ken Pu
Research Co-supervisor	Dr. Heidar Davoudi
Examining Committee Member	Dr. Faisal Qureshi
Thesis Examiner	Dr. Amirali Abari

The above committee determined that the thesis is acceptable in form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate during an oral examination. A signed copy of the Certificate of Approval is available from the School of Graduate and Postdoctoral Studies.

Abstract

This work outlines a method for performing natural language tasks as part of a relational framework. Utilizing features of PostgreSQL as a relational database and its extensibility to allow for word embedding without leaving the relational database. This system can be extended to incorporate several natural language processing(NLP) techniques, such as latent Dirichlet allocations(LDA) or modern models, such as BERT. The combination of NLP and relational operations allows for extracting data from and analyzing text in the same interface used for general data analysis. This combination allows for gathering richer information from existing sources and makes it all available from one standard interface. The declarative nature of SQL allows for more ad-hoc application of NLP techniques. Two case studies using the DBLP dataset demonstrate this integration's power. Building an LDA model, augmenting the topic labels for greater descriptiveness, and applying preexisting models for semantic analysis.

Keywords: Query language, database, natural language proccessing, embedding vectors, text processing

Authors's Declaration

[illegible]	Andrei Stoica

Statement of Contribution

Part of the work described in Chapter 3 has been published as:

A. Stoica, K. Q. Pu and H. Davoudi, ”NLP Relational Queries and Its Application,” 2020 IEEE 21st International Conference on Information Reuse and Integration for Data Science (IRI), 2020, pp. 395-398, doi: 10.1109/IRI49571.2020.00064.

I performed the majority of design, writing and testing of the code as well as the writing of the manuscript.

Acknowledgements

I want to thank everyone that has helped me along this long journey, specifically, Thanks to my supervisors, Dr. Ken Pu and Dr. Kourosh Davoudi, who lent their expertise and guidance over the years. Thank you for working with me to get my thesis over the finish line despite their packed schedules during the last summer.

I also want to thank Dr. Faisal Qureshi and Dr. Amirali Abari for serving on the supervisory committee and as the examiner. Thank you for pushing me to ensure that this thesis is as high quality as possible.

Thank you to Dr. Jeremy Bradbury, who served as co-supervisor for my undergraduate thesis alongside Dr. Pu. That thesis gave a small introduction to research and academic writing.

Thank you to my super supportive girlfriend, that encouraged me and pushed me to finish when I was ready to give up. Sorry for all the late nights I had to spend working on this. Thanks for inspiring me and pushing me to be the best version of myself.

Lastly, I'd like to thank everyone else not specifically named yet. All the fantastic people I met at Ontario Tech, from professors to students. Every that has inspired me to improve, that I have looked up to, that have driven me to learn more or work harder.

Contents

List of Figures

List of Listings

Chapter 1

Introduction

1.1 Motivation

In the modern digital world, we witness an ever-growing data volume. In the pre-World Wide Web (WWW) days, most data sources were financial institutions, governments and businesses. Such datasets are mostly in relational format and were stored in relational database management systems (RDBMS), such as PostgreSQL[1].

However, since the popularization of social media and the WWW, text data has become the dominant source of data. Text is distinct from traditional relational data in that it is unstructured and has historically been challenging to be understood by computer programs. Nevertheless, it has been recognized that text data contains valuable information that can be leveraged to gain insight into users and their social environments.

Both relational and text data have been the subject of active research on algorithms to understand data. Understanding data is a fundamental challenge in the field of data science. Using various tools, data scientists' goal is to gain a deep un-

[1] www.postgresql.org/

derstanding of relational or textual data and extract useful information that can be used for data-driven decision support.

Example 1 *To illustrate our motivation, we will consider a real-life scenario. Consider the task of selecting suitable reviewers for a given research grant proposal. Imagine that there is a proposal document consisting of some structured information:*

- *A set of proposal investigators*
- *The research institutions of the investigators*
- *date of the proposal*

The proposal also has unstructured textual data, which is the proposal document containing the title, abstract, and detailed description of the research program.

The task is to compute a list of suitable candidates as reviewers who must satisfy the following selection criteria:

1. *The reviewer must not be a coauthor with any of the proposal investigators in the last five years.*
2. *The reviewer cannot be from the same research institution as any of the investigators.*
3. *The reviewer must belong to a list of recognized universities.*
4. *The reviewer's area of expertise must coincide with the topic of the proposed research program.*

■

Criteria 1-3 of Example 1 can be addressed by structured queries based on the relational data of the proposal and researcher database. However, criteria 4 requires semantic understanding of textual data.

The above example illustrates the exact type of query that any existing solution cannot easily solve. When one criterion requires semantic understanding of text, it falls out of the scope of relational queries. The first three criteria are a natural fit for the declarative nature of query languages such as SQL. The last query would require semantic understanding of the proposal text. This information is outside the reach of current relational database operation.

This thesis focuses on designing and implementing a system for semantic analysis of text that is compatible with the declarative interface of relational databases. This work would allow for complicated queries such as those in example 1

1.2 Existing Methods To Analysis Text And Relational Data

The existing methods to analyze relational and textual data are segmented into different domains:

1.2.1 Text Embedding Vectors

Many techniques have been developed over the last decade for computer programs to help better understand unstructured text corpora.

Word level embedding:

With the advocates of neural networks and deep learning, we have ever increasingly more powerful text embedding methods such as Word2Vec [27], FastText [8] and

GloVE [34].

These word embeddings are computed in such a way as to capture the semantics of each word. This can be leveraged to satisfy Criteria 4 of the problem outlined in example 1. Although these embeddings are at the word level, there are numerous ways to compute a single vector for a larger piece of text. The simplest is an average or a weighted average using TF-IDF. There have also been adaptations of Word2Vec that embed an additional vector for each paragraph during training. [21] This vector is embedded into the same space as the token-level embeddings.

Utilizing these vectors can be helpful in finding a set of reviewers for our original query in example 1. It is possible to measure the average distance between the vectors representing our proposal and potential reviewers' publications, potentially utilizing a dataset such as DBLP. The nearest neighbours can be considered the closest matches in terms of research focus.

Sentence level embedding:

More recently, Bidirectional Encoder Representation Transformers (BERT) [12] is a language model that can generate embedding vectors of sentences. It takes advantage of the power of transformers to represent connections between tokens from different parts of the text. Transformers give it a better understanding of the semantic connections of words than other language models.

BERT is a neural network that contains many hidden layers. One of these layers is an embedding layer. The embedding layer can be used in place of the document vectors computed from the sum of individual token-level embeddings.

Document level embedding:

Latent Dirichlet Allocation (LDA) [6] generates vectors from text documents, with each weight being the score for a particular topic. LDA is an early version of a text embedding technique.

The topic vectors generated by LDA do not have the same high dimensionality as those generated by the embedding techniques discussed above. Additionally, this is applied only at the document level. Each element of the vector represents a latent topic. Thus if a reviewer's works cover the same latent topics, they can also be considered a good candidate.

Collectively, these methods allow us to convert text data into high-dimensional vectors suitable for downstream data analysis.

1.2.2 Structured Queries

Databases:

Relational databases (RDBMS) have always supported powerful query capabilities via the Structured Query Language (SQL). SQL has many features: the storage engine scales very well, from small to large datasets. SQL queries can be easily composed to support ad-hoc exploration tasks. Also, SQL has many features designed to support real-time and data stream query workloads. The historical limitations of SQLhave been that it only applies to relational data and does not integrate well with unstructured data and foreign data sources.

Relational queries are a straightforward way to resolve the first 3 criteria of example 1.

1. The reviewer must not be a coauthor with any of the proposal investigators in the last five years:

```
SELECT unique(unnest(authors)
FROM publication
EXCEPT
SELECT unique(unnest(authors))
FROM publication
```

```
WHERE authors && investigators
AND year > proposal_date - 5
```

2. The reviewer cannot be from the same research institution as any of the investigators.

```
SELECT person
FROM affiliation
WHERE NOT institute in investigator_institutes
```

3. The reviewer must belong to a list of recognized universities

```
SELECT person
FROM affiliation
JOIN recognized_university
ON affiliation.institute = recognized_university.name
```

The intersection of these queries will give us the list of eligible candidates for the position of reviewer.

Map Reduce Systems:

Google introduced map-reduce [11], which started the era of Big Data Analytics. Systems such as Hive[2] and Spark[3] are designed to run on large clusters of computers running as map-reduce type of jobs. However, these systems are exclusively suitable for large-scale computing clusters and do not adapt well to real-time and ad-hoc scenarios.

Data Analytics with Python:

Data science encompasses the analysis of both text and relational data. When the data volume is reasonable, many data scientists rely on a generic programming language such as Python and a data analytic library such as Pandas [4]. Libraries such

[2]hive.apache.org/
[3]spark.apache.org/
[4]pandas.pydata.org

as Pandas provide convenient and flexible API to slice and dice the data sets to fit the needs. Furthermore, when using a general-purpose programming language such as Python, it is efficient to integrate with text processing libraries. For example, the Python library Gensim [40] implements the LDA algorithm. Python also has libraries for Word2Vec, FastText and BERT text embeddings. All of these libraries can be used together with Pandas to perform analysis of structured data analysis within Python. But a serious problem with Python is limited scalability: the run-time environment is fundamentally limited to the Python interpreter.

Other systems, such as PostgreSQL and Spark, can be configured to scale horizontally by augmenting the underlying hardware and computer clusters.

1.2.3 Visualization

There are ample amount of tools and libraries available to support data visualization. Some are Web-based (e.g. d3.js [5]) generating interactive data visualizations in the browser. Python libraries such as matplotlib [6] can be used together with Pandas to generate static graphical files that visualize the analytical results. There are also commercial products (e.g. Tableau [7]) that focus on data visualization and query analytics.

Regardless of choice for data visualization, the integration with the data analytic query language and data processing platform always remains a challenge.

1.2.4 Data Analytic Pipelines

Given the segmented systems and libraries for text and relational data analysis, data scientists are faced with the problem of building data analytic pipelines. A pipeline

[5]d3js.org
[6]matplotlib.org
[7]www.tableau.com

is a collection of loosely integrated data processing modules that perform individual tasks but are connected by input-output data flow.

A typical pipeline involves different stages, typically referred to as *extraction*, *transformation*, and *load*. An ETL pipeline performs **extraction** of data from its raw sources, and then **transforms** the raw data into a manageable format. The transformations for text will involve tokenization and vectorization into embedding vectors. The final stage of an ETL pipeline is **loading** the transformed data into the analytical platform. This final destination for the data may be a PostgreSQL database or a Python Pandas dataframe. Depending on the choice of the platform, one would use different query languages to perform the analysis.

Some shortcomings of ETL pipelines are:

- Any pipeline can be complex and costly to maintain when the raw data sources are updated frequently.
- Since transformation happens before load, any design change in the transformation stage would require a complete overhaul load of the new transformed data. This additional cost is a direct result that the analytical platform is separate from data transformation.

Systems such as Apache Airflow [8] are designed to manage complex pipelines.

One solution to the selection criteria posed in Example 1 is to build an ELT pipeline to solve all 4 constraints simultaneously. Here we can leverage the power of a tool such as Airflow to execute our queries and then aggregate the information when they finish executing. This sort of solution would require us to build a solution for each of the technologies used separately and then the Airflow configuration to manage them.

[8]airflow.apache.org

For large companies with infrastructure and engineers dedicated to setting up and maintaining such pipelines, it is not problematic to implement it in such a way. When it comes to smaller tasks such as that proposed in Example 1, they often are solved by a single person or small team. These types of problems benefit from an ad-hoc environment because a query such as that might only run once a year or a couple of times a year, and the criteria might change from year to year.

1.3 Utilizing Existing RDBMS Features

The benefit of utilizing SQL for an end-to-end system is that there are existing elements in the SQL feature set for the following:

- Easy to compose elements
- Data pipeline
- Data freshness
- Data sharing
- Access control
- Data integrity

Each step in a traditional data transformation pipeline can be expressed in any given language, taking advantage of the wide range of data science libraries available on that given platform. RDBMS have been solutions for data transformations for longer than a language such as Python. As such, it has amassed an abundance of features for this task which are going unused in the current landscape of data pipelines.

SQL as a language is very well suited for expressing data transformation. The language has many features that enable it to make rigorous procedural code simple and easy to write while also supporting complex nested queries.

Nested queries and common table expressions(CTE) enable complex data transformations to be represented as more minor transformations. Then these small queries can be composed to build complicated transformations. Accessing supplementary tables or other relational information can be done anytime, as more information is required. Sub-queries/CTEs or other tables can be combined using joins, a built-in way of combining related information.

Modern RDBMS support numerous complex datatypes out of the box or with existing extensions. This is advantages for supplementing an NLP query with auxiliary information.

Here is a non-comprehensive list of datatypes that PostgreSQL supports or has an extension to add support:

- JSON
- XML
- Geospatial
- Timeseries

For even more complicated queries, SQL allows for building views. These views can be materialized or virtual; both can be queried as though they are a table. Where a materialized view is constructed ahead of time and has a cached version stored on disk that is accessed, and a virtual view is constructed at query time.

These virtual views can be constructed to represent a step in the transformation pipeline and can be queried as is to enable exploration of that step. The views can also be used later in the pipeline as they are needed. Building a pipeline using views allows us to compose each step one at a time. Then they can be easily inspected to ensure that the result is as intended because they can be queried as though they are ordinary tables. When the result of a previous step in the pipeline is needed further down the road, they again can be accessed as though the data is stored in a table.

Materialized views are an option for steps which require more computation time. These views are computed ahead of time and stored on disk. The results are computed with the cached version on disk at query time. Any updates to the original data are incorporated only if a view is refreshed.

Data freshness can be ensured even when using a materialized view by utilizing triggers on the previous tables. Once new data is inserted on the source tables, a trigger function can start the recomputation of downstream views. The timing of these computations has to be managed so that the database is not overloaded trying to recompute materialized views. This recomputation is especially dangerous when source tables receive data frequently.

RDBMS have features to enable data access across multiple servers, and specifically, PostgreSQL has an extension to enable this functionality called postgres_fdw. The foreign data wrapper extension enables access to tables from external instances. By importing a foreign schema, a table can be queried as though the table exists in the same database but be in a database on a server in a different part of the world. This functionality enables the spread of data over multiple servers and the ability to aggregate it into views and tables as though it is one large database. Apart from the networking overhead, these views can be built as though the original data existed together in the same database.

The foreign data wrapper takes care of access control as well. When declaring the wrapper for a new server, a mapping of user accounts must be specified to enable access. Mappings provide the login credentials for each user when connecting to the remote database.

In RDBMS, particularly PostgreSQL, transaction control and data locking are enforced by default. These systems provide locks for data at the table, row, page and database level. The built-in commands in PostgreSQL acquire the appropriate locks automatically, and these locks can also be acquired manually if necessary.

Locking can ensure that the source data is not modified during the training and testing of multiple models. If the underlying data changes between two models while varying hyperparameters, the two are no longer comparable. Locking can be used in combination with materialized views for tables that constantly receive new entries or updates.

A materialized view can be used to create a snapshot of a table. Then this view can be locked so that view is not modified between training sessions. This will enable access to the original table and does not slow data intake for a system with constant data flow. However, this still enables a consistent dataset for training and testing.

Random sampling is also possible within the RDBMS. PostgreSQL provides the *random* function that can be used in conjunction with an order by to randomize the table. To retrieve N random samples from a PostgreSQL table

```sql
SELECT *
FROM foo
ORDER BY random()
LIMIT <N>;
```

1.4 Contribution

This thesis aims to enable semantic natural language queries to be mixed with traditional relational queries to solve complex queries such as that in example 1. SQL provides a rich feature set that enables answering criteria 1-3, but there are still features not yet implemented that enable it to identify the solution for criteria 4. Thus it can act as a unified platform for these mixed queries.

To allow for these semantic comparisons of text data inside of PostgreSQL. The following is implemented in PL/pgSQL:

- Linear algebra implementation to support vector distance
  - Vector arithmetic
  - Norm (L1 and L2)
  - Distance measures
- Vector Aggregation
  - Mean

Additionally, pl/Pythonu functions are written to handle the model:

- Inferencing
- Training

This integration would empower us to solve the four criteria of example 1 with a single query.

Assuming we have a table of embedding vectors for each author's works:

```sql
SELECT name, dist(document_vector(proposal), embedding) as dist
FROM candidate
JOIN author_embedding USING (name)
ORDER BY distance DESC
```

Here, *candidate* is the intersections of the queries that satisfy criteria 1-3. *Candidate* can be expressed as a series of common table expressions.

The usefulness of this is demonstrated with two case studies into the DBLP dataset. The first describes how this integration can be used to enable the generation of higher quality labels for LDA topics. The technique utilizes the ability of this solution to reconstruct mappings from tokens back to words dynamically. This generates human-understandable multi-word phrases to take the place of the standard token labels. The second utilizes a generally available FastText model to provide semantic insight into the publication that can be applied to individual authors, institutes and venues in academia.

1.5 Organization

The layout of the thesis is as follows.

Chapter 2 discusses the many facets of text analysis. This chapter includes the concept of topic modelling and other text embedding techniques. It will cover the steps required to perform text analysis in vector space and some benefits of working inside a vector space. It will also introduce a dataset that will reappear as an example throughout this work.

Next, Chapter 3 introduces the minimal code necessary for performing the case studies. The code first goes over the vector arithmetic, including; addition/subtraction, multiplication/division, norm and aggregation. This chapter also includes code for training and loading models in PostgreSQL. The code for model handling only covers the models and methods used in the case study; however, these can be extended to any word embedding model that was not mentioned or does not yet exist.

The case studies will be covered in Chapter 4. The first case study covers building an LDA model and improving upon the standard topic labelling process. The process of building a topic model in the rigid world of relational databases will be discussed. It will cover new techniques that are possible with access to relational data and topic modelling. This will show the unique power of accessing relational data and topic modelling in a single interface. It will show how topic modelling adds more functionality to the existing SQL toolbox and allows for deeper analysis of text fields inside relational databases. The second case study will cover loading a pre-trained model in SQL and utilizing it to perform deep analysis of the same dataset. Here it is demonstrated how these methods can apply to new techniques and updated as better embedding techniques emerge.

Chapter 5 will cover an evaluation of performing such analysis with a relational database engine. It compares the analytical functions introduced in Chapter 3 to similar implementations that utilize Python and Pandas. As well as explore the possibility of further performance improvements from PostgreSQL's built-in indexing features.

Lastly, chapter 6 will discuss some related works. These technologies not only provide some of the features we rely on but also those that have not been directly implemented in this work can be incorporated similarly. This chapter will cover recent advances in Natural Language Processing that can provide even greater power to understanding the text stored in many databases.

Chapter 2

Background

2.1 Text in Data Science

Today's online world is full of data displayed and, more importantly, stored in text form. From news articles to Twitter and other micro/macro blogging platforms and even user reviews on sites such as Amazon or Yelp, there is an abundance of data being stored and exchanged in text form. Text is quite challenging to analyze. Even with the many techniques that have arisen over the past few decades, which are quite effective, it requires multiple forms of analysis to extract sufficient information. This is due to the fact that each algorithm is specialized in analyzing one aspect of the text. From topic analysis to sentiment analysis or, more recently, the creation of general purpose feature vectors to be used in a myriad of other tasks. The results of each task are often aggregated into larger datasets to give an overall picture of the text. There is much information to be extracted from text; however, complex transformations are required to access it.

One such type of text data analysis that can be performed is to infer the sentiment for user reviews on Yelp or Amazon [39]. The score of the review can be used to train this model to identify positive or negative sentiment. This model can then be used to infer the sentiment of other pieces of text, such as Twitter or another type of blog.

We can take Twitter as a case study of what types of information can be embedded in text. Firstly, today Twitter has a character limit of 280 characters per tweet, up from the original 140 character limit with which it launched. This limit means a single tweet can only span one or maybe two separate topics, unlike a longer news article or blog, which can cover many topics. Usually, more extensive articles, such as from traditional news organizations, focus on one topic but may cover it from multiple perspectives and how it connects to other aspects of life. Some more complex topics are hierarchical and can even encompass many smaller topics. This complexity could mean that in one part of the article, the sentiment can be very positive, another less so, and in some parts, it could be negative. The sentiment can shift as the article navigates the many aspects of a complex topic or as it transitions from one topic to another. Since tweets are very short and thus can only cover a limited number of topics, we can infer the sentiment from the tweet as a whole. We can be reliably certain that this sentiment clearly pertains to one topic. Thus we can infer that user **X** feels a certain way toward topic **Y**.

A specific use case of this technology would be to track how Twitter users feel about public policy or a candidate. Similarly, companies can use this to track public opinion about them or their product. User behaviour on Twitter is such that a tweet highlights its topic through the use of hashtags. This behaviour facilitates the easy search for tweets about the same subject on the site. We can take advantage of this and attribute the tweet's sentiment to that hashtag. A real-world example of this is during the Canadian political cycle when there is a federal election, and we see an increase in the use of the hashtag *#CdnPoli* as the general public discusses candidates. Similar trends happen with provincial elections, such as with the *#OnPoli*, here, in

Ontario. These tweets can also include other hashtags that can identify the topic more precisely, such as a single party, candidate or a piece of policy.

These sorts of systems require a lot of engineering work to build and maintain due to the segmented toolset. Separate tools have been built for each step of the processing pipeline. Additionally, there needs to be a system for controlling the pipeline.

2.2 Structured Data in Data Science

In contrast to unstructured data in the form of text, structured data does not require much processing to extract its value. From semi-structured data forms, such as JSON or XML, to completely structured data, such as that stored in a relational database management system(RDBMS), the structured nature of this data means there is less processing required to extract information. Take the description of a student as an example:

> Mathew is a grade 9 student. He has brown hair and blue eyes. He got a 95 in math and an 80 in English.

Extracting information from this data requires a lot of processing and knowledge of the English language. However, if it is stored in a more structured form such as JSON, it is easier to compare students and compute statistics if a standardized form is used for all records. Data in a structured record is also preferred because it is naturally more space efficient. For example, numbers are stored in the correct format; thus, decimal numbers do not need to be stored as several characters. Moreover, there are no stop words or ambiguity, which would need many more characters to clarify. Since the beginning, data science has focused on storing data in a dense and standardized format so multiple records can be compared and for which statistics can be computed easily. The JSON equivalent is provided in Figure 2.1.

```
{
        "name": "Mathew",
        "current grade": 9,
        "marks": {
                "math": 95,
            "English": 80
    }
}
```

Figure 2.1: JSON Student Record

This sort of standardization of stored data has been around since the early days of computers, even a simple CSV file. Even though a CSV does not have as much flexibility as a JSON file or the power of RDBMS, it offered a more standardized way to store records. A student record can be stored in the form **Name**, **Class**, **Mark**. These versions of student records were the only way to reasonably store and compare records at the dawn of data science.

Until relatively recently, it was unrealistic for a computer to extract the necessary information from a block of text like our original example. Like with the field of machine learning, there has been work done in the field of text mining for decades. However, we only recently have gained access to hardware powerful enough to analyze text on a grand scale. Along with hardware advancements recently, we have also seen an increase in text mining algorithms' sophistication, making the field's real-world applications possible. Thus data science has historically focused on storing and analyzing data in structured data.

One such form of structured data comes in the form of relational databases. First appearing in 1970 in a paper form IBM [9], RDBMS have established themselves as the industry standard for data storage. There are many RDBMS offerings from big companies:

- DB2 from IBM [1]

[1]https://www.ibm.com/products/db2-database

- Oracle Database[2]
- Microsoft SQL Server [3]

There are also many open source or free offerings such as:

- PostgreSQL [4]
- SQLite [5].

Most of the popular offerings today are based on the SQL language. They feature a standard syntax and some variation in some of their functionality. More recently, NoSQL databases have gained popularity. They provide similar storage and retrieval functionality but sacrifice the analysis features for better integration with native data structures. For example MongoDB [6] stores JavaScript Objects like documents(JSON).

RDBMS has become such an industry standard that all three major cloud providers have some managed, scalable implementation of this technology. Google has Cloud SQL[7] for a fully managed version of MySQL, PostgreSQL or SQL Server, and they offer Bigtable [8] and Firestore [9] as NoSQL options for their customers. Microsoft offers Azure SQL Database [10] and Cosmos DB [11] for SQL and NoSQL applications, respectively. Amazon has Aurora [12] and RDS [13] for traditional SQL applications and offers DynamoDB[14] as a NoSQL option.

[2]https://oracle.com/database
[3]https://www.microsoft.com/en-ca/sql-server/sql-server-2019
[4]https://www.postgresql.org/
[5]https://www.sqlite.org
[6]https://www.mongodb.com/
[7]https://cloud.google.com/sql
[8]https://cloud.google.com/bigtable
[9]https://cloud.google.com/firestore
[10]https://azure.microsoft.com/en-ca/services/sql-database
[11]https://azure.microsoft.com/en-ca/services/cosmos-db
[12]https://aws.amazon.com/rds/aurora
[13]https://aws.amazon.com/rds
[14]https://aws.amazon.com/dynamodb

RDBMS and NLP are great data analytics tools and helpful in today's digital environment. A single toolchain to analyze complex data would simplify the technology stack in many use cases and allow context switches between the two modes of analysis to happen more seamlessly. This integration requires work at the query language and the model level. An integration such as this would allow an even better understanding of some semi-structured data where long-form text is embedded and contains critical information.

2.3 Motivation in Topic Modeling

Over the years, documents have been organized according to various categorization systems. These systems have always relied on human labels to identify the overarching theme of each text. Before digital storage, most systems require labelling and physically storing the document in a specific spot. These systems aid humans in finding a document that covers the topic in which they are interested. Before the digital age, many used only one label for each document because a single document can only be in one place at a time. This sort of system could be problematic because any document that is complex enough to be of interest would naturally cover multiple topics. In the digital age, we have adapted these systems to use multiple keywords to aid in the discoverability of documents. Nevertheless, these labels are generated ahead of time by humans.

For categorization and searchability, a document can be assigned single labels. These labels capture the general topic discussed in the text. Take, for example, a library. Most use the Dewey Decimal System to organize their collection of books into an indexed system. Allowing people to find books on topics of interest easily. For greater usability, this system organizes the labels into a hierarchal structure that starts with broad topics like natural science and mathematics, language or technology. Then allows a user to drill down to more specific topics such as mathematics or geometry, which are both sub-topics of natural science and mathematics. Some sub-topics can be even more specific then geometry.

Another system for labelling documents is to use multiple keywords. Amazon uses multiple keywords to augment their categorization. Multiple keywords improve the browsing experience through this more modern interface where the user can move from one category to another in seconds instead of walking across a room or building. Amazon's system also augments this system with keywords for user preference to better match user searches with relevant results. It further matches keywords of potential search results with user preference keywords to increase the relevance of top results to the user.

With systems like these, document labels are still produced by humans. Only after a human takes time to read a document can it be organized by these systems and indexed for discovery by others. With topic modelling, we can generate human-readable labels like previous systems. Additionally, one document can belong to multiple topics, and it can model how much each document covers each topic. This model forms a high-dimensional representation of each document. Documents can then compare documents based on what topics they cover. We can find similar documents and recommend them in applications such as Amazon or in the case of searching for academic papers that cover similar topics.

2.4 Text Mining

Text mining is a form of computer-aided analysis of text for the means of extracting useful insight and patterns. It can employ techniques such as information retrieval, natural language processing (NLP) and other machine learning (ML) algorithm. It can be viewed from 2 different points of view. Firstly, it can be viewed as simple information extraction, retrieving facts from a body of text. Secondly, it can be viewed as text analysis, where algorithms and methods borrowed from other fields, such as ML and statistics, are utilized to discover patterns in the text. [15]

Going back to the example of tweets. There will certainly be some tweets that do not identify their topics precisely using hashtags and some that do not use hashtags at all. Other techniques in natural language processing(NLP) can assist in identifying the major topics in a body of text, broadly called topic analysis. There are classical statistical approaches such as Latent Semantic Analysis(LSA) and Latent Dirichlet Allocation(LDA). Also, there are approaches based on machine learning or even deep learning.

Lexalytics [15] is a company that specializes in extracting patterns and insights from comments on social media, reviews or other text documents. They have many high-profile customers, such as Oracle and Microsoft.

Topic modelling is one technique that exists in the realm of NLP. It draws statistical correlations between text and a set of topics. Many different algorithms can accomplish this goal. Latent Dirichlet Allocation is the most versatile and commonly used of these techniques. In the time since LDA was first proposed, many variations of the original algorithm have appeared that further improve performance in specific tasks, such as ToPMine [14] which adds the ability to directly mine phrases from the text and lda2vec [31] which embeds both word vectors and topic vectors in the same vector space so to aid in creating topic labels.

A few techniques are standard across the field of text mining. Generally, these Techniques can be applied to text as a whole. They can range from the removal of

[15]https://www.lexalytics.com/

extraneous words or combining words that have similar enough meanings. Extraneous words can be something like the word 'the.' These words carry little importance to the subject or meaning of the sentence but complicate the algorithms we use significantly. Nonetheless, they are part of natural language, and we can use some simple techniques to clean up the raw text and improve the results of subsequent algorithms we employ for text analysis. Other times we can simplify multiple words with similar meanings to a single token. One example of this is as simple as removing any capitalization. In this case, subsequent algorithms will treat all instances of a word the same. If we did not, these algorithms would treat an instance of a word differently if it appears at the beginning of the sentence just because its first character is different. We could take more complicated steps, such as collapsing all words into their root.

One technique that can be employed is token normalization, where we transform the text to remove redundant or uninteresting/unimportant information. One step in this process is called stemming, where words are replaced by their roots. Take the following sentence as an example:

Saly ran to the markets.

In this example both **ran** and **markets** would be replaced by their root words **run** and **market**. Another step would be to remove stop words, which are extremely common in natural language but do not convey much information. For example: **the**, **a**, **as** and even the word **and** are all stop words required for proper English grammar but do not convey much information. If normalization was applied to the sentence above, it would be transformed into:

Saly run market

While this reduces the granularity of the information stored in the original text, it also helps draw connections between a word's plural, past and present forms. Without these steps to preprocess the text, many existing algorithms would perceive each form of a word as a distinct token. Some algorithms can be affected negatively by preprocessing the text in this manner.

Having separate tokens for each of these tenses can have its advantages. Such as reducing the number of tokens and the complexity for subsequent algorithms. It can help LDA by finding clearer correlations between tokens and thus building better topics in the process. Some other algorithms in the NLP toolbox are robust enough to handle raw text or even at some in-between stage of normalization. Some of these tools are best suited for only specific preprocessing steps. One must apply these preprocessing steps with knowledge of what algorithm will be applied further down the line. There is no one size fits all solution.

There are also many algorithms that can embed words into high-dimensional space. The embedding process can be accomplished by using a classical statistical approach or through the use of modern machine learning techniques.

GloVe [34] is one such statistical approach to word embedding. The system builds its embedded representation based on the word-word co-occurrence matrix. The matrix contains the number of times word **i**, and word **j** occur in the same context. The embeddings are then computed using these co-occurrence statistics.

Other approaches use a shallow neural network that infers this co-occurrence information by scanning the text using a sliding window like Word2Vec [29]. These models create representations of words in a high-dimensional space by training a shallow neural network to predict the word at the center of the sliding window based on the sum of the words that appear around it or in the reverse order. The two models are called continuous bag of words(CBOW) or skip-gram. Both model types have their drawbacks and advantages. Another variation called FastText [8] breaks each word into parts and embeds each part individually. Using sub-word embeddings can help improve accuracy for rarely used words as well as eliminate the need to stem words ahead of time. This method is able to capture the meaning of the root and its suffixes and prefixes separately. More recently, BERT [12] further improved on these techniques through the use of Bidirectional Transformers in the NN structure.

Another area where these technologies can be applied is scientific publications. A scientific conference usually focuses on a single topic or a small set of interconnected topics. Similarly, a single author naturally also focuses on a limited set of topics. The topics covered by a conference or author can be identified using technologies such as LDA. Naturally, these topics shift over time as new technologies emerge. Topic modelling can not only be used to identify topics covered by a conference but it can be adapted to identify the change in topics as new technologies are introduced. The distribution of topics will naturally drift from one year to another when comparing the LDA topic distributions from 2 different years.

2.5 From Documents to Vectors

Working with documents in text form is cumbersome. In its most natural form, documents are stored and represented as a sequence of characters. Mathematical reasoning in this form is immensely difficult. There are methods to compare documents directly in this form. However, a tiny change in the characters can sometimes significantly change the meaning. Take, for example, changing from single to plural form or from present to past tense. The change in semantic meaning from those modifications is minimal. For example, ”I put the apple in the basket” versus ”I put the apples in the basket.” The meaning of the sentences is virtually identical, apart from the number of apples involved in the action. On the other hand, other minor changes at the character level could have a significant impact on the semantic meaning. Take, for example, adding a word. Starting with the sentence "A book on rolling rocks." By changing it to "A book on rock and roll," the book's topic has changed drastically, and it has gone from the topic of geology to music. The semantic difference is extreme, but it was brought on by a slight change to the characters/words present in the text itself.

These limitations could be improved by representing documents as a bag of words and even further improved by first tokenizing the words via stemming and storing these new tokens. Stemming eliminates the different tenses of each word. Storing just the root of each word means a sentence in the present tense and the past tense that describes the same action would be represented by the same tokens.

Representing our document using a bag of words allows us to compare documents more easily. We can encode them as one-hot vectors where each dimension would represent the presence of a specific token in the document. The downside of this method is that our dimensionality is equal to the number of tokens present in the entire dataset, and these vectors are enormous and very sparse. Another downside is that when we want to add a new document with a previously unencountered token, we have to increase the dimensionality of all of our vectors to be even larger. This method involves large vectors representing documents, theoretically reaching and exceeding the English dictionary. For large datasets, it is impractical to represent documents using this method.

The field of text mining has provided algorithms that can transform documents into vectors with lower dimensionality. These algorithms use word-word co-occurrence statistics to generate vector representations of documents. Each dimension in these vectors represents one topic. Since the number of topics discussed in a set of documents is much smaller than the number of distinct words, this method significantly reduces our dimensionality. Allowing for more efficient storage of documents and reducing computation power when comparing them.

Using this method, changes at the character level do not necessarily have a direct impact on the final vector representation. Since the topic vector is generated from word co-occurrence statistics, they capture the semantic meaning of tokens. So if two words are generally used in the same context, these algorithms will represent them as having similar meanings. One example of this would be the use of plural forms, even without stemming the context in which singular and plural forms of words are similar. We are afforded much numerical flexibility by taking advantage of these algorithms and representing documents as topic vectors. Differences between documents are numerically quantifiable. In a vector space, many existing methods measure the differences or distance between documents can be applied. We can compare two documents by taking the cosine similarity of their topic vectors. Since we are dealing with these documents in vector space, we can easily apply geometric analysis to find patterns and extract more information. For example, performing dimensionality reductions to be able to allow for plotting in 2D and inspecting visually. We can even perform k-means clustering to find small clusters of closely related documents.

2.6 Topics in Vector Space

Mapping our topics into vector space allows for advanced mathematical analysis. PCA or SVD could be used to investigate the relationship between documents or clustering can be applied to identify groupings of similar documents.

PCA and SVD are great dimensionality reduction techniques that reduce large dimension rectangular matrices. These techniques help make sense of the data for large datasets with over ten topics and identify how documents are clustered in the higher dimensional space.

Furthermore, clustering algorithms such as K-means are great tools for finding patterns and closely related documents in vector spaces such as this one. For example, this can be used to find communities of academics in DBLP that publish on closely related topics and potential collaborators for future papers.

These vectors are also handy for machine learning(ML). A document's topic vector can be used as pre-computed features to be fed into an ML model. Combining this topic information with existing feature vectors can improve performance on tasks such as classification.

PostgreSQL has native support for vectors. It has the capability to can create an array data type, and this array can store the entire vector. It is possible to have multiple models and store vectors from each model in separate columns or tables. Another way to store these vectors is to create a new table for each model, and each element in the vector would get stored in a separate column.

Storing these vectors relationally has many advantages. Firstly, we have convenient access to them when analyzing documents. For example, if we have a dataset of dated documents and we want to look at how many documents are published per month, we can tack on the aggregation of the topic vectors to get an idea of what is discussed each month. Secondly, if information from that database is accessed through a REST API, there is less work to access this new information about the documents. Instead of having a separate NLP pipeline and importing the vectors back into the database after processing the documents, the entire pipeline is now inside PostgreSQL. With this implementation, more engineering time can be shifted to the API endpoint.

2.7 Text Analysis in a Vector Space

Working with text in vector space allows us to apply pre-existing techniques directly. Text in the form of an ordered list of tokens/words is not conducive to this sort of analysis. Many algorithms exist for comparing and analyzing easily computed vectors and have existing implementations in many libraries for languages such as Python. K-mean clustering, dimensionality reduction and principal component analysis(PCA), to name a few.

Clustering can be used to identify closely related documents, and it can locate pockets in our high-dimensional representation of densely populated documents. All documents in one of these clusters have a similar distribution of topics. Intuitively, the papers published at a single conference would form one of these clusters as they should generally cover similar topics with slight variation.

K-Means clustering is the most popular among these clustering algorithms. As such, it already has pre-existing implementations in many libraries such as scikit learn[16]. Implementations of this algorithm exist for PostgreSQL as well[17]. It is implemented as a user-defined function that is compatible with our topic vectors out of the box. Without any additional work, This library can be leveraged to perform clustering inside of PostgreSQL. Allowing for the combination of information gained by clustering with other metadata and information from the original text using standard SQL such as simple JOIN.

Representing our documents as high-dimensional topic vectors captures a lot of essential data to distinguish the general topic of each document and compare documents. It can be cumbersome for humans to rationalize in this high-dimensional space. When a person inspects the vectors for a set of documents, looking at the raw

[16]https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
[17]https://github.com/umitanuki/kmeans-postgresql

vectors is impractical. There already exist many algorithms and solutions for this problem. Many plotting and dimensionality reduction techniques can aid in human analysis of high-dimensional datasets like the one we are generating.

When trying to visualize high dimensional data such as this, it often helps to project points into a lower dimensional space. Displaying a high-dimensional plot on a 2D screen is problematic, and it can become tough to read and understand, even in three dimensions. Many projections into lower dimensional space exist and vary in scope and goal. Multi-Dimensional Scaling(MDS) is a projection that attempts to maintain the exact pair-wise distances as in the original high-dimensional space. This projection helps preserve relative positions and thus can be very useful in plotting and visualizations.

Similarly, primary component analysis(PCA) is another algorithm for projecting data into a lower dimensional space. It differs from MDS because it aims to identify the planes whose projections contain the most information. The first of which contains the most deviation in position, and each subsequent dimension would contain decreasing amounts of information. This projection can aid in visualizing high-dimensional data by identifying the two dimensions that contain the most information and projecting onto them to preserve the greatest amount of information about variation when creating a 2D plot of our data.

2.8 Latent Dirichlet Allocation

Using LDA in the implementation allows us to take advantage of the many implementations and variants of LDA that exists for specific tasks. This allows for much flexibility in the applications of our solution. LDA also has numerous non text-based applications [16]. LDA has existed for decades, so there has been much research into its possible applications and extensions.

LDA is a generative model that generates words in a document. These words are generated from multinomial distributions whose parameters are described by the topic weights of the document at hand.

To generate a document in LDA, first, the number of words must be selected, **N**. Then specifying the topic mixture for the documents. Finally, each word is generated by selecting a topic according to the distribution of the topic mixture and then selecting a word from that topic's distribution of words. Repeat that last step until **N** words have been generated for the document.

These parameters of our model, topic mixtures and word distributions for each topic are the topic weights in vector space.

2.9 Word2vec and fastText

Word2vec [27] was a seminal paper for vector representations of text. Since this work was introduced, several works have extended it for numerous tasks and adapted its technology for other fields. Doc2vec [21]s is a more general model that allows for learning document vectors during the same training phase.

These models are shallow neural networks. There are two general model architectures: Continuous Bag of Words(CBOW) and Skip-gram. The representation of each word is learned based on the context in which it appears. The two architectures differ in how the training happens. With a CBOW architecture, the context around the word is used as input. The vectors of these words are summed to obtain a prediction. Whereas with the Skip-gram model, the reverse is true. The current word is used as input, and the neural network predicts the words that appear in its context.

FastText [8] is another evolution of word2vec's simple architecture. The general model is based on the Skip-gram model; however, the difference is in the method used to model the text. The fastText model breaks words down into pieces and is able to recognize the meanings of smaller sets of characters, such as prefixes and suffixes, that modify a word's meaning in the same ways every time they appear.

The main difference is that instead of word2vec's single vector for each word, fastText will break down each word into character n-grams and a unique token for the entire word. The vector representation would then be the average of the vectors for all character n-grams and the unique token representing the entire word. This modification is advantageous in many ways.

First, because each character n-gram has its own embedding, it can capture the meanings of smaller sequences of characters. Separate embeddings are helpful when dealing with suffixes and prefixes. The n-grams can be embedded separately, and their unique meaning is captured. When a word's vector is generated, all of the n-gram's vectors will add some component to the overall vector for that word.

Secondly, splitting words into n-grams enables better representations of words that do not frequently appear in the training corpus. Since the vector representation is computed by averaging the vector embeddings of its n-grams, The meaning of its component n-grams can capture some of the meaning of the word itself. For example, if the training set does not have the word 'homeless' more than a few times, word2vec would struggle to capture the word's true meaning. However, fastText can take advantage of the fact that two n-grams, ['home', 'less'], appear that carry most of the words meaning. Both are common in the English language. There is a good chance that the suffix 'less' has a good embedding while 'home' is a common English word. This can extend to numerous words that are unlikely to appear many times, even in a broad dataset.

Lastly, unlike word2vec this system allows for vector representations of words that did not appear in the original training text. This is limited by the fact that some n-grams might not have been seen either. The quality of the vector for the out-of-vocabulary(OOV) word goes up with the percentage of n-grams for which it can generate vectors.

2.10 BERT

BERT [12] is different from both word2vec and fastText. It is a language model. Its primary goal is not to create vector embeddings of text. However, BERT feature vectors can be used as text embeddings. Normally, a language model is trained by predicting the next token for a given sequence. BERT makes use of new ML models that allow it to better model correlations between words than a traditional language model.

The power of BERT comes from the use of transformers which introduce attention. This allows it to make connections between tokens encountered in different positions. Usually, this sort of connection would only be able to be made between the last few tokens that came before the current position due to the way normal models scan through sequences. Previous language models scan left to right, only giving them access to information about what tokens have passed.

BERT also uses a new training methodology to allow the model to learn more about relations between tokens. Instead of predicting the next word, BERT uses a masked language model. Words are removed from a text, and the model predicts the missing words. A masked language model allows BERT to gather information from the context before that position, but more importantly, it can learn from the context after that token. A second task is performed. to improve performance further, where it scans two sentences and predicts whether the second sentence is the sentence that immediately follows the first in the original text. This new method of training allows BERT to draw better correlations between all words inside of a sentence, and because of the second stage, it can learn about connections between sentences. This next sentence prediction benefits some NLP tasks, such as question answering.

After training BERT, there are two ways in which it can be used for NLP tasks. First, we can take the inner weights as a feature vector and use this for our intended task. Using these weights can be effective, but it depends on how we create our feature vector. The original paper presents and compares six methods for creating such feature vectors from BERT. [12] All six methods are applied to the CoNLL-2003 Named Entity Recognition(NER) task. [42] From the results, utilizing only the embedding layer performed the worst with only a 91 F1 score on the Dev set. While the other methods used some combination of the last Four hidden layers of the network, either picking one of the last two or concatenating/summing the last four and one test with a weighted sum of all 12 layers. All but the embedding layer had F1 scores of $\tilde{9}5$. With the concatenation of the last Four layers got an F1 score of 96.1. These feature vectors almost matched the performance reached by fine-tuning the model for a downstream task.

Fine-tuning this model for the NER dataset reached an F1 score of 96.4 on the Dev dataset. The process of fine-tuning involves plugging the new input and output into the model and adding a new final layer depending on the task. The sentence pair features can be used for tasks such as question/answer. The token representations can be used for token-level tasks. While a special token is used to identify each piece of text, which can be used for document-level tasks such as classification or sentiment.

Both BERT and fastText are state-of-the-art technologies for their respective goals. While this thesis implements NLP operators in the form of topic modelling using LDA, The general framework for implementing NLP operators extends to these technologies as well. The Gensim library, which was used for its implementation of LDA in this thesis, also includes an implementation of word2vec, doc2vec and fastText. With only minor modifications to the functions and tables, we could just as easily use these technologies to generate document vectors for clustering or similarity. There is already work done to implement top clustering algorithms in SQL [32,44] as well as distance calculations [35]. BERT would require the use of Tensorflow[18] but it can create feature vectors that can be used in a similar way to those of fastText or LDA. However, with the power of using BERT and Tensorflow, we can fine-tune it for more specific tasks. This framework would be applicable to both cases.

2.11 DBLP as an example

DBLP[19] is a bibliography of computer science publications. It will be used as an example for this workflow throughout this thesis. The overall dataset contains computer science publications of all types. There are journal articles, conference proceedings, workshop papers, books, thesis and even informal publications. Only conference proceedings and workshop papers have been used to limit the scope of the analysis. Even with this limited subset of publications, there are still 3,057,306 records in the dataset that will be utilized. Each record has the following structure:

id, **title**, **author**, **year**, **pages**, **booktitle**

In which: *id* is a unique string for each publication; *title* is the title of the publication; *author* is a list of authors separated by "::"; *year* is the year of the publication; *pages* is the page count; *booktitle* is the name of the conference or workshop.

[18]www.tensorflow.org
[19]https://dblp.org/

For example, the record for a paper:

conf/iri/StoicaPD20, NLP Relational Queries and Its Application., Andrei Stoica::Ken Q. Pu::Heidar Davoudi, 2020, 4, IRI

There is no information about the contents of the paper itself. With the absence of keyword information, there is little that can be done in a traditional relational interface to find other related papers to this one. It is possible to find papers that have also been written by one of these authors or published at the same conference. There would have greater flexibility in what papers can be identified as similar if it were possible to extract more information from the title.

The easiest step is using full-text search to search for keywords in the title. This feature is already built into most SQL software. However, there is more information that is out of reach still. We still cannot measure how closely related two documents are, just that they share a keyword found within their title.

With the system outlined in this thesis, it is possible to extract more information about the title. There is support for comparing documents where most information about their relations to one another is trapped within text fields. The system outlined here will be able to answer questions like "Is the topics covered by a particular venue or author changing more rapidly in the past years?", "Are venues in similar fields converging on the same topics or finding their own niches over time?" or "Which of my colleagues do my publications most align with?"

These sorts of questions are answered through a combination of data that can be extracted from text as well as traditional relational data. The use of LDA topic vector or FastText sentence vector, in combination with the year and venue information, makes an in-database implementation of NLP tasks great at solving these sorts of queries.

Chapter 3

PostgreSQL Extension

This section outlines the implementation of the necessary PostgreSQL Functions.

3.1 Element-wise Vector Operations

Following is a list of all vector functions that operate on the vector elements. These functions are implemented in pl/PGSQL, including vector arithmetic and distance functions. For each, we will provide a mathematical definition and a simple example.

For the running example, we will consider a simple relational table containing coloured points in 3D.

All vectors will be represented by float arrays, `FLOAT[]`, in PostgreSQL. The table schema is:

```
CREATE TABLE points3d(
  id INTEGER PRIMARY KEY,
  color ENUM('red', 'green', 'blue'),
  v FLOAT[]
)
```

3.1.1 Addition

Element-wise vector addition is implemented as the `arr_add(a, b)` as shown in Listing 1. For completeness, we will provide the straightforward definition:

$$\texttt{arr\_add}(a, b)[i] = a[i] + b[i]$$

pl/PGSQL provides loop iteration which can be used to implement the given function.

Listing 1 Addition of 2 vectors

```
CREATE FUNCTION arr_add(a FLOAT[], b FLOAT[])
RETURNS FLOAT[]
AS $$
DECLARE
    n INTEGER;
    c FLOAT[];
BEGIN
    n = LEAST(array_length(a, 1), array_length(b, 1));
    FOR i in 1..n LOOP
        c[i] = a[i] + b[i];
    END LOOP;
    RETURN c;
END;
$$ LANGUAGE plpgsql;
```

Example 2 *Recall the table of 3D points in* `point3d`*. Suppose that we want to redefine the origin of the points to be* `(-1, -1, -1)`*, which involves adding the vector* `(1, 1, 1)` *to* `point3d.v`*. This can be done using the function SQL statement with the* `arr_add`*.*

```
UPDATE points3d
SET v = add_arr(v, array[1, 1, 1]);
```

■

Element-wise addition is necessary for the later computation of a mean vector. In the Python ecosystem This functionality is provided by the Numpy package, where the default `+` operator is overridden to add support for addition of the `np.array` type. In Listing 1, the function for element-wise addition of 2 vectors is defined. Each element i in the new array **c** is the sum of the i^{th} elements of arrays **a** and **b**.

3.1.2 Subtraction

The subtraction function is given in Listing 2.

Listing 2 Subtraction between 2 vectors

```
CREATE FUNCTION arr_sub(a FLOAT[], b FLOAT[])
RETURNS FLOAT[]
AS $$
DECLARE
    n INTEGER;
    c FLOAT[];
BEGIN
    n = LEAST(array_length(a, 1), array_length(b, 1));
    FOR i in 1..n LOOP
        c[i] = a[i] - b[i];
    END LOOP;
    RETURN c;
END;
$$ LANGUAGE plpgsql;
```

Euclidean distance requires the element-wise subtraction between 2 vectors to calculate the vector between them. Numpy once again provides this functionality in the Python ecosystem. Numpy overrides the python `-` operator for the `np.array` datatype. Listing 2, contains the pl/PGSQL function for this functionality. Each element i in the new array **c** is the difference between the i^{th}elements of arrays **a** and **b**.

3.1.3 Multiplication

The element-wise multiplication of vectors is defined as:

$$\texttt{arr\_mul}(a, b)[i] = a[i] * b[i]$$

The pl/PGSQL implementation is given in Listing 3.

Listing 3 Element wise multiplication between two vectors

```
CREATE FUNCTION arr_mul(a FLOAT[], b FLOAT[])
RETURNS FLOAT[]
AS $$
DECLARE
    n INTEGER;
    c FLOAT[];
BEGIN
    n = LEAST(array_length(a, 1), array_length(b, 1));
    FOR i in 1..n LOOP
        c[i] = a[i] * b[i];
    END LOOP;
    RETURN c;
END;
$$ LANGUAGE plpgsql;
```

Example 3 *Suppose we wish to compute a new set of vectors by scaling the x and y axes of points in* `point3d` *by factors of 2 and 3, respectively. Furthermore, we do not want to modify the original vectors. This can be done by the following:*

```
SELECT id, arr_mul(array[2, 3, 1], points3d.v)
FROM points3d;
```

■

In the python ecosystem Numpy adds element-wise multiplication of arrays by overriding the `*` operator for `np.array`. In listing 3, a pl/PGSQL function that returns an array `c`. The i^{th} element of **c** is the product of the i^{th} elements of arrays **a** and **a**.

3.1.4 Division

Element-wise division is shown in Listing **??**.

Listing 4 Element wise division between two vectors

```sql
CREATE FUNCTION arr_div(a FLOAT[], b FLOAT[])
RETURNS FLOAT[]
AS $$
DECLARE
    n INTEGER;
    c FLOAT[];
BEGIN
    n = LEAST(array_length(a, 1), array_length(b, 1));
    FOR i in 1..n LOOP
        c[i] = a[i] / b[i];
    END LOOP;
    RETURN c;
END;
$$ LANGUAGE plpgsql;
```

To complement the multiplication for scaling division is also supported. The function is included to make scaling down easier. This function can be used to reduce a vector by 50%. Instead of multiplying by an array containing 0.5s, dividing by an array containing all 2s can be performed. In a Python environment, Numpy also overrides the `/` operator for element-wise division of the `np.array`. In listing 4, the i^{th} element of the array `c` is the quotient of the i^{th} element of arrays **a** and **b**.

3.1.5 Norm

We have provided two functions to compute the norm of vectors. The function `arr_l1_norm` computes the L1-norm of vectors as defined as:

$$\texttt{arr\_l1\_norm}(a) = \sum_i |a[i]|$$

Similarly, the L2 norm is computed by `arr_l2_norm` defined as:

$$\texttt{arr\_l1\_norm}(a) = \sum_i |a[i]|^2$$

Their implementations are shown in Listing 5 and Listing 6 respectively.

Listing 5 L1 norm of a vector

```
CREATE FUNCTION arr_l1_norm(a FLOAT[])
RETURNS FLOAT
AS $$
DECLARE
    n INTEGER;
    x FLOAT;
BEGIN
    n = array_length(a, 1);
    x = 0;
    FOR i in 1..n LOOP
        x = x + ABS(a[i]);
    END LOOP;
    RETURN x;
END;
$$ LANGUAGE plpgsql;
```

Listing 6 L2 norm of a vector

```sql
CREATE FUNCTION arr_l2_norm(a FLOAT[])
RETURNS FLOAT
AS $$
DECLARE
    n INTEGER;
    x FLOAT;
BEGIN
    n = array_length(a, 1);
    x = 0;
    FOR i in 1..n LOOP
        x = x + a[i] * a[i];
    END LOOP;
    RETURN sqrt(x);
END;
$$ LANGUAGE plpgsql;
```

These are just a subset of the norm functions provided by Numpy as part of its linear algebra library for the Python ecosystem. However, for this use case described in this thesis, there is no need to implement more than these two. Norm is required for later computation of similarity.

Listing 5, contains the implementation of L1 norm in pl/PGSQL. In that function, `x` is returned as the sum of the absolute values of each element of the input array `a`.

Listing 6, contains the implementation of L2 norm in pl/PGSQL. In that function, `x` is returned as the sum of the square of each element of the input array `a`.

Example 4 *Consider the running example of 3D points; we utilize the norm functions to identify the longest vector.*

```sql
SELECT id
FROM points3d
WHERE arr_l2_norm(v) = (
  select MAX(arr_l2_norm(S.v)) FROM points3d S
)
```

This query demonstrates the flexibility and expressiveness naturally obtained from using SQL. The nested query in the `WHERE` *clause allows us to filter vectors by their norms.* ■

3.1.6 Dot Product

The dot product is implemented as `arr_dot`, and defined as:

$$\texttt{arr\_dot}(a, b) = \sum_i a[i] * b[i]$$

The implementation is shown in Listing 7.

Listing 7 Dot product between 2 vectors

```
CREATE FUNCTION arr_dot(a FLOAT[], b FLOAT[])
RETURNS FLOAT
AS $$
DECLARE
    n INTEGER;
    x FLOAT;
BEGIN
    n = LEAST(array_length(a, 1), array_length(b, 1));
    x = 0;
    FOR i in 1..n LOOP
        x = x + a[i] * b[i];
    END LOOP;
    RETURN x;
END;
$$ LANGUAGE plpgsql;
```

Like the norm functions listed earlier in this section, dot product will be required by the similarity functions listed later. Numpy has a `np.dot` function that does dot product for `np.array` in the Python ecosystem. In listing 7, the returned value `x` is the sum of the products between the i^{th} elements of input arrays `a` and `b`.

3.1.7 Cosine Similarity

Listing 8 Cosine Similarity between 2 vectors

```
CREATE FUNCTION arr_cos_sim(a FLOAT[], b FLOAT[])
RETURNS FLOAT
AS $$
DECLARE
    x FLOAT;
BEGIN
    x = arr_dot(a, b) / arr_l2_norm(a) / arr_l2_norm(b);
    RETURN x;
END;
$$ LANGUAGE plpgsql;
```

Cosine similarity is a measure of similarity based on the angular distance between two vectors. Due to it being a measure of angular distance, it ignores the length of the vectors. In listing 8, the returned value of the function is the dot product of the two vectors divided by the L2 norm of **a** and the L2 norm of **b**.

Example 5 *Suppose we wish to identify pairs of distinct vectors which are the most similar in* `point3d` *measured by their cosine similarity.*

```
SELECT S.id, T.id
FROM points3d S, points3d T
WHERE arr_cos_sim(S.v, T.v) = (
    SELECT MIN(arr_cos_sim(X.v, Y.v))
    FROM points3d X, points3d Y
    WHERE X.id != Y.id
)
```

■

3.1.8 Euclidean distance

Another way of measuring closeness between two vectors is their Euclidean distance, as defined as:

$$\texttt{arr\_euclidean\_dist}(a, b) = \|a - b\|$$

as implemented by Listing 9.

Listing 9 Euclidean distance between 2 vectors

```
CREATE FUNCTION arr_euclidean_dist(a FLOAT[], b FLOAT[])
RETURNS FLOAT
AS $$
DECLARE
    x FLOAT;
BEGIN
    x = arr_l2_norm(arr_sub(a, b));
    RETURN x;
END;
$$ LANGUAGE plpgsql;
```

3.2 Matrix Multiplication

Many data science analysis requires matrices. We can naturally represent matrices as nested arrays of floats: `FLOAT[][]` in PostgreSQL.

3.2.1 Matrix-Vector Multiplication

Suppose we have $y = Mx$ where M is a matrix of shape (m, n) and x is a vector of shape n. The result y is a m-dimensional vector given by:

$$y[i] = \sum_{j=1}^{n} A[i][j] * x[j]$$

The function `mat_arr_mul` implements the matrix-vector multiplication in pl/PGSQL.

Listing 10 Matrix-vector multiplication

```
CREATE FUNCTION mat_arr_mul(mat FLOAT[][], x FLOAT[])
RETURNS FLOAT[]
AS $$
DECLARE
    y FLOAT[];
    m int;
    n int;
BEGIN
    m := array_length(mat, 1);
    n := array_length(mat, 2);

    FOR i in 1..m LOOP
        y[i] = 0;
        FOR j in 1..n LOOP
            y[i] = y[i] + mat[i][j] * x[j];
        END LOOP;
    END LOOP;

    RETURN y;
END;
$$ LANGUAGE plpgsql;
```

Listing 10 contains a simplified version of matmul for matrix and array multiplication. While matrix-vector multiplication could also be accomplished with the matmul function from Listing 11, The interface for matmul requires the array to be modified to a 2-dimensional array of shape 1xn. The value returned is also a 2-dimensional array. The function in Listing 10 is compatible with flat arrays as input and returns flat arrays.

Example 6 *Suppose we wish to perform the following projection.*

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \to \begin{bmatrix} x+y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

This can be done in SQL with the help of the `mat_arr_mul` *as:*

```
SELECT id, mat_arr_mul(
    array[[1, 1, 0]
          [0, 0, 1]],
    v)
FROM points3d;
```

■

3.2.2 Matrix-Matrix Multiplication

Often, matrices are used to describe linear operators in the context of data science. Thus, compositions of such operators can be computed as matrix-matrix multiplication.

The function `matmul` implements matrix-matrix multiplication as shown in Listing 11.

Listing 11 Matrix-matrix multiplication

```
CREATE FUNCTION matmul(a FLOAT[][], b FLOAT[][])
RETURNS FLOAT[]
AS $$
DECLARE
    c FLOAT[][];
    m int;
    n int;
    p int;
BEGIN
    m := array_length(a, 1);
    n := array_length(a, 2);
    p := array_length(b, 2);

    ASSERT array_length(a, 2) = array_length(b, 1);

    c := array_fill(0, ARRAY[m, p]);

    FOR i in 1..m LOOP
        FOR j in 1..n LOOP
            FOR k in 1..p LOOP
                c[i][k] = c[i][k] + a[i][j] * b[j][k];
            END LOOP;
        END LOOP;
    END LOOP;

    RETURN c;
END;
$$ LANGUAGE plpgsql;
```

First, the dimensions of the 2 arrays are verified to be compatible for multiplication. The output array is initialized to all 0s. The output array is fill such that $c_{ik} = \sum_{j=1}^{n} a_{ij}b_{jk}$

Example 7 *Consider the projection operator as defined in the previous example:*

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow \begin{bmatrix} x+y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Suppose we wish to apply a 2D rotation after the projection. The 2D rotation is represented by the rotational matrix defined by:

$$\begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}$$

The composition of the two operators can be computed using pl/PGSQL functions as:

```
matmul(array[
  [cos(theta), sin(theta)],
  [-sin(theta), cos(theta)]
], array[
    [1, 1, 0],
    [0, 0, 1]
])
```

■

3.3 Aggregation Functions

The following listings contain aggregation functions. Each aggregation function is made up of 3 parts: First, there is a state. It is created with the command `CREATE TYPE`. Second is a function to update that state for each vector. Then there is a final function to convert that state into the final value that will be returned. Each of these is required when creating an aggregation using `CREATE AGGREGATE`

3.3.1 Mean

Listing 12 Mean of a set of vectors

```sql
CREATE TYPE state_t AS (total float[], num int);
CREATE FUNCTION update_state(
    state state_t,
    arr float[],
    out result state_t)
AS $$
BEGIN
    if state is null then
        SELECT arr, 1
        INTO result;
    else
        SELECT arr_add(state.total, arr), state.num + 1
        INTO result;
    end if;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION avg_finalfunc(state state_t)
RETURNS float[]
AS $$
DECLARE
    result float[];
BEGIN
    FOR i IN 1..array_length(state.total, 1) LOOP
        result[i] = state.total[i] / state.num;
    END LOOP;
    RETURN result;
END;
$$ LANGUAGE plpgsql;

CREATE AGGREGATE arr_mean(float[]) (
    sfunc = update_state,
    stype = state_t,
    finalfunc = avg_finalfunc
);
```

For a set of vectors, such as those representing one author's publications, computing the mean vector allows for the use of one vector and a distance measure, such as those listed above, to compare sets. This functionality can be implemented in Python using Numpy arrays and the mean function or with Pandas and its built-in mean function.

In pl/PGSQL, it is implemented as an aggregation function. Made up of three parts; a type, an update state function, and a final function. The code is featured in listing 12.

The state is made up of an array and a counter. The array stores the sum of vectors, and the counter stores the number of vectors.

The updated functions initializes the state on the first call to contain the first array and the count to 1. The new vector is added to the state upon subsequent calls, and the counter is incremented.

The final function is called when all vectors are consumed. It returns an array that is made up of the i^{th} element of the state vector divided by the count.

Example 8 *Suppose we wish to compute the centroids of each of the colors. This can be accomplished by a SQL group-by query with the defined aggregation function* `arr_mean`*.*

```
SELECT color, arr_mean(v)
FROM points3d
GROUP BY color;
```

■

3.3.2 Variance

Listing 13 Variance on a set of vectors

```
CREATE TYPE var_state_t AS (
    ss float[], s float[], num int
);
CREATE FUNCTION update_var_state(
    state var_state_t, arr float[], out result var_state_t
)
AS $$
BEGIN
    if state is null then
        SELECT arr_mul(arr, arr), arr, 1
        INTO result;
    else
        SELECT arr_add(state.ss, arr_mul(arr, arr)),
            arr_add(state.s, arr),
            state.num + 1
        INTO result;
    end if;
END;
$$ LANGUAGE plpgsql;
CREATE FUNCTION final_var_func(state var_state_t)
RETURNS float[]
AS $$
DECLARE
    sq_avg FLOAT[];
    avg  FLOAT[];
BEGIN
    FOR i in 1..array_length(state.ss, 1) LOOP
        sq_avg[i] = state.ss[i] / state.num;
        avg[i] = state.s[i] / state.num;
    END LOOP;
    RETURN arr_sub(sq_avg, arr_mul(avg,avg));
END;
$$ LANGUAGE plpgsql;
CREATE AGGREGATE arr_var(float[]) (
    sfunc = update_var_state,
    stype = var_state_t,
    finalfunc = final_var_func
)
```

3.4 Model Handling

In this section, we will present a collection of functions that perform model based machine learning tasks including model loading, inference, and training.

We will demonstrate their usage in a detailed case study involving an end-to-end workflow of text analysis in SQL and various NLP models that are trained from data and pre-trained, as shown in Chapter 4.

3.4.1 Model Inference

This section will cover the implementation of the pl/pythonu functions for loading and utilizing existing models.

Listing 14 Inference of existing models

```
CREATE FUNCTION inference(document TEXT, model TEXT, model_type TEXT)
returns real[]
AS $$

    if model_type == 'fasttext':
        import fasttext

        if not SD.get(model):
            SD[model] = fasttext.load_model(model)

        model_inf = SD[model].get_sentence_vector
        model_input = document.strip()

    elif model_type == 'lda':
        from gensim.models import LdaModel
        from gensim.corpora.dictionary import Dictionary

        if not SD.get(model):
            dict = model + '.dct'
            SD[model] = LdaModel.load(model)
            SD[dict] = Dictionary.load(dict)

        model_inf = SD[model].get_document_topics
        model_input = SD[dict].doc2bow(document)

    else:
        raise Exception('model type not supported')

  return model_inf(model_input)
$$ LANGUAGE plpython3u;
```

Listing 14 contains the code for loading and inferencing existing models. It is built for LDA via the Gensim package and fasttext models. Each model has different requirements for their respective API, but generally, they will have some inference function and preprocessing steps. Therefore, the function branches to load and preprocess separately. To save time, first, the model is attempted to be loaded from local context `SD` if it is not already in memory, it is loaded from disk. Finally, the model inference function is called with the preprocessed input.

For fasttext models, the inference function can consume the raw string and return a vector. There is no need for other preprocessing steps outside of removing any extra whitespace.

For LDA models, we must also load an accompanying Dictionary object. This object is used to transform the string into a bag of words, and it can split a string on whitespace. However, it does not perform any stemming or stop word removal. If any of those steps were performed when the model and dictionary were created, they must take place before the document is consumed by `lookup()`.

This function can be updated to add additional support for models. However, this is the minimum set of models needed for the case studies in chapter 5.

3.4.2 Model Training

Listing 15 Training new LDA models

```
CREATE FUNCTION train_lda(documents TEXT[], model TEXT, topics INTEGER)
returns BOOLEAN
AS $$

    from gensim.models import LdaModel
    from gensim.corpora.dictionary import Dictionary

    docs = [doc.split() for doc in documents]
    dict = Dictionary(docs)
    corpus = [common_dictionary.doc2bow(doc) for doc in docs]

    trained_model = LdaModel(corpus, num_topics=topics)
    trained_model.save(model)

    return True
$$ LANGUAGE plpython3u;
```

The function for training an LDA model is featured in listing 15. Here the input documents are once again a single string of whitespace-separated tokens. Any preprocessing should be applied before passing it to this function. The parameter model is the path to save the model, which will be used to recall it later via the function in listing 14. The last parameter specifies the number of topics for the model.

Chapter 4

Case Studies

4.1 Topic Modeling with LDA

4.1.1 Functional characterization of topic modeling

This subsection provides a functional characterization of topic modelling. The types and functions involved in topic modelling will be defined.

Each document has at least the following attributes:

- id(x): the unique identifier of the document x.
- text(x): the entire textual representation of the document x.

Thus, the document type is defined as:

$$\mathbf{Doc} = \mathbf{ID} \times \mathbf{Text}$$

A lexical analyzer is used to map the raw textual representation into a sequence of words in a given dictionary.

$$\text{lex} : \textbf{Text} \rightarrow \textbf{list}\,\langle\textbf{Word}\rangle$$

In practice, different words can share the same root, and for the purpose of topic analysis, we want to normalize words into their roots, known as the word stems. The word stem captures the topical meaning, while the rest of a word conveys information such as negation and tense. It is these word stems that are going to be referred to as tokens of the documents. This is a type of text normalization called stemming. There are many stemming algorithms, and all can be described as implementations of the following function:

$$\text{stem} : \textbf{Word} \rightarrow \textbf{Token}$$

The stem function can be extended to apply to a whole document by composing stem with lex:

$$\text{stem} \circ \text{lex} : \textbf{Text} \rightarrow \textbf{list}\,\langle\textbf{Token}\rangle$$

A topic model is trained using a collection of documents. Given a set number of topics, **k**, the topic model assigns a vector of size **k** to each token. The model is able to infer topic vectors for documents using these token vectors, even those it has not seen during training.

A topic vector is a discrete probability distribution over k topics. Where $\vec{t_i}$ is the probability of an entity belonging to the i topic.

$$\textbf{TopicVector} = \mathbb{R}^k$$

The inference function of a topic model has the following signature:

$$\text{infer} : \textbf{list}\ \langle\textbf{Token}\rangle \rightarrow \textbf{TopicVector}$$

It transforms a list of tokens representing the document into a topic vector representation of the same document.

4.2 Topic model functions in SQL

In order to smoothly integrate topic modelling into a relational database system (RDBMS), SQL native data types and functions need to be created that mirror these given in Section 4.1.1 along with those already implemented in Chapter 3

Modern RDBMS engines (e.g. PostgreSQL) offer full programming constructs and data type definitions to implement the topic model functions.

For example, creating a function that performs lexical analysis to split our text into an array of words. Here it is possible can take advantage of PostgreSQL's built-in functions to simplify the implementation. The builtin function `regexp_split_to_array` can be used to split on whitespace and return an array of words.

```
CREATE FUNCTION lex(string TEXT)
RETURNS TABLE(word WORD)
LANGUAGE plpgsql AS
$$
    DECLARE
        arr TEXT[] := regexp_split_to_array(string, '[ ]+');
    BEGIN
        RETURN arr;
    END
$$;
```

PostgreSQL comes with full-text search functionality, which includes stemming from normalizing different variations of the same keyword root. This feature can be used to implement `stem`.

```
CREATE FUNCTION stem(text TEXT)
RETURNS TOKEN LANGUAGE SQL AS
$$
    WITH T AS (
        SELECT lexeme
        FROM unnest(to_tsvector(text))
        WHERE array_length(positions, 1) > 0
        ORDER BY positions[1]
    )
    SELECT array_agg(lexeme) AS stem FROM T
$$;
```

Topic model training and inferencing are done with the functions listed in Chapter 3. The trained model is saved to disk with the train_lda function in Listing 15. Then the model can be inferences with the inference in Listing 14.

4.2.1 Integrated topic modeling workflow in SQL

The motivation for integrating topic modelling functions into SQL is to improve the end-to-end workflow for text analysis that involves topic modelling. In this section, it is demonstrated how basic topic modelling is possible with an RDBMS. Furthermore, it is also demonstrated how topic modelling can be tightly integrated within a larger context of a data processing workflow.

Consider a database that has a relation of documents in their text form.

The relation may be a simple table:

```
doc_text(doc_id INTEGER, text TEXT)
```

This system allows for the creation of a view for these documents in token form. Taking advantage of `stem` and `lex` to create this view.

```
CREATE VIEW doc_tokens(doc_id INTEGER, tokens TEXT) AS
SELECT doc_id, array_to_string(stem(lex(text)), ' ', ' ')
FROM doc_text
```

A topic model can be built on the tokenized documents.

```
\set num_topics 20;
WITH docs AS {
SELECT array_agg(tokens)
FROM doc_tokens
}
SELECT train_lda(docs, '/data/dblp_20.lda', num_topics);
```

Once the model is built, extracting the topic vectors of specific documents is possible.

```
SELECT
    id,
    inference(tokens, '/data/dblp_20.lda', 'lda') as topic_vec
FROM doc_tokens
LIMIT 3;
```

The results are similar to the following table. We are only showing the weights of four topics for three of the documents in the database.

```
             id               |               topic_vec
------------------------------+--------------------------------
 conf/3dgis/AbdallaAT06       |     {0.00, 0.00 ... 0.00, 0.00}
 conf/3dgis/AchilleF06        |     {0.25, 0.00 ... 0.00, 0.00}
 conf/3dgis/Al-HanbaliASAB06 |     {0.00, 0.00 ... 0.00, 0.00}
```

RDBMS offers a vast collection of query facilities that can now be used to generate the topic vectors of documents of interest. It is possible to perform keyword search for documents of interest and even filter based on some extra metadata that is stored alongside the document. Such as author or publication in the case of academic documents such as those in DBLP.

Suppose that we want to perform keyword search over the document text. This search can be done efficiently using full-text search features in PostgreSQL. Then the inferencing step can be applied to provide the top vectors for the topic document that match our criteria.

The following example performs topic model inferencing on documents that contains the keywords deep, learning and the phrase topic modeling.

```
SELECT doc_id, inference(tokens, '/data/dblp_20.lda', 'lda')
FROM doc_text JOIN doc_tokens USING (doc_id)
WHERE 'deep & learning & topic <-> modeling'::tsquery
        @@ doc_text::tsvector
```

Suppose that there are additional metadata on the documents scattered in tables.

```
doc_title(doc_id INTEGER, title TEXT)
doc_author(doc_id INTEGER, author TEXT)
```

```
WITH T(booktitle, topic_vec)
AS (
         SELECT
        booktitle,
        inference(tokens, '/data/dblp_20.lda', 'lda')
        FROM
        doc_info NATURAL JOIN doc_tokens
                 NATURAL JOIN doc_title
        WHERE 'deep & learning'::tsquery @@ title::tsvector
   )
SELECT booktitle, arr_mean(topic_vec)
FROM T
```

The query above is performing several data processing:

- Topic vectors are computed for the
- A keyword search “deep learning” based on the title

- Computing the aggregate mean topic vectors over different booktitles of publications.

```
           booktitle              |           topic_vec
----------------------------------+--------------------------------
 AAAI                             |    {0.06, 0.00 ... 0.00, 0.00}
 AAMAS                            |    {0.25, 0.00 ... 0.00, 0.00}
 ACIVS                            |    {0.49, 0.05 ... 0.00, 0.05}
 ACM Multimedia                   |    {0.00, 0.00 ... 0.00, 0.00}
 AIM                              |    {0.19, 0.10 ... 0.00, 0.00}
 AIME                             |    {0.00, 0.00 ... 0.22, 0.11}
 AIST (Supplement)                |    {0.00, 0.00 ... 0.00, 0.00}
 Allerton                         |    {0.00, 0.00 ... 0.00, 0.00}
 Bildverarbeitung für die Medizin |    {0.00, 0.00 ... 0.00, 0.00}
 CBMS                             |    {0.82, 0.00 ... 0.00, 0.10}
```

So far, this has only included the most cursory outline of how this work provides the framework to perform integrated text analysis rapidly. In subsequent sections, we will focus on several key aspects.

- It will be demonstrated that SQL can be used to generate topic labels for each topic.
- RDBMS provides data indexes that can significantly accelerate text analytical tasks.

4.2.2 Novel Technique For Generating Semantic Labels of Topics

Consider the task of generating meaningful topic labels so that each `topic_`n has semantic meaning associated with them. This improves human understanding of our generated topics.

The topic model represents each topic as a linear combination of tokens. This representation is very cumbersome when thinking about the central theme of each topic.

For this reason, we generate text labels to represent topics. Increasing understanding when looking at them.

Constructing a labelling function:

$$\text{label} : \textbf{Topic} \to \textbf{Text}$$

Traditional topic model assigns topic scores to single tokens:

$$\text{score} : \textbf{Topic} \times \textbf{Token} \to \mathbf{R}$$

This induces a ranking of the tokens with respect to each topic. This is given as the number of tokens with a greater relevance score to the topic t.

$$\text{rank}(x|t) = |\{y \in \textbf{Token} : \text{score}(t, y) \geq \text{score}(t, x)\}|$$

Example 9 Here is an example of the top scores of some tokens for a topic associated with a model for the DBLP dataset.

```
token   | score    | rank |
--------+----------+------|
imag    | 0.051038 |  1   |
detect  | 0.042553 |  2   |
use     | 0.037227 |  3   |
multi   | 0.037086 |  4   |
base    | 0.036398 |  5   |
featur  | 0.023689 |  6   |
segment | 0.017756 |  7   |
track   | 0.015788 |  8   |
model   | 0.011814 |  9   |
method  | 0.010413 | 10   |
```

■

The most straightforward (and commonly used) labelling technique is to use the most relevant n tokens.

$$\text{label}_{\text{naive}}(t : \textbf{Topic}) = \text{string}\left(\{x \in \textbf{Token} : \text{rank}(x|t) \leq n\}\right)$$

Example 10 The naive label for the above example with $n = 5$ will generate the following semantic label: `imag detect use multi base`. ■

From the example above, it is possible to see two immediate weaknesses:

- Individual tokens do not correspond to words in the title due to the process of stemming. For example `image` has been stemmed to the token `imag`.
- Many semantic concepts are expressed as multi-word phrases in the text. However, the lexical analyzer has broken the text down into words (which are stemmed into tokens). Unfortunately, at the word level, much of the semantic meaning is lost.

The problem of generating high-quality semantic labels is best solved using both topic modelling tools (topic scores and ranks of tokens) and database processing (statistical reversal of stemming and lexical analysis). Using both toolsets together allows for the combined power to generate more descriptive and easily interpretable labels.

The first step is to find the most meaningful token bigrams.

```
CREATE FUNCTION candidate_bigrams(min_rank INTEGER)
RETURNS TABLE
LANGUAGE SQL
AS $$
    SELECT topic, T1.token AS token_1, T2.token AS token_2
    FROM token_scores T1 JOIN token_scores T2 USING (topic)
    WHERE T1.rank > min_rank
          AND T2.rank > min_rank
$$;
```

The candidate token bigrams are identified using the `candidate_bigrams` function. It finds the top-ranked tokens for each topic and generates all possible pairs from these tokens. Next, it utilizes the database's full-text search facility to count the number of times token bigrams appear in the (stemmed) titles of documents.

```
CREATE VIEW token_bigram_freq AS
  SELECT DISTINCT
      token_1,
      token_2,
      COUNT(DISTINCT doc_id) AS freq
  FROM candidate_bigrams(50)
  JOIN doc_text
  WHERE stem(lex(title))::tsvector @@ to_tsquery(token_1 || '<->' || token_2)
```

This function uses `token_bigram_freq` to identify phrases (as token bigrams) with strong semantic meaning (e.g. `imag segment`). However, in practice, it returns too many false positives: bigrams that contain frequently occurring tokens that do not generally form high-quality phrases. From the DBLP dataset, it returns false negatives such as `use feature`. Even though it is not a phrase, it frequently appears in titles as ... `using feature based` .... It is possible to draw from information retrieval literature and design a scoring function that promotes phrase scores while also penalizing token frequencies to combat this issue. This is exactly how the well-known term-frequence-inverse-document-frequency (TFIDF) score works.

First, identify the token frequency.

```
CREATE VIEW token_freq AS
    WITH T AS (
        SELECT doc_id, UNNEST(stem(lex(title))) AS token
        FROM doc_text
    )
    SELECT token, COUNT(DISTINCT doc_id) AS freq
    FROM T
```

Finally, put it together to generate the semantic labels for the topics.

```
CREATE VIEW bigram_scores AS
    SELECT
        token_1,
        token_2,
        log(B.freq + 1) / log(T1.freq + 1) / log(T2.freq + 1) as score
    FROM token_bigram_freq B
        JOIN token_freq T1 ON B.token_1 = T1.token
        JOIN token_freq T2 ON B.token_2 = T2.token
```

Now, to rank the candidate token bigrams using the computed bigram scores.

```
CREATE VIEW topic_token_bigrams AS
    SELECT
        row_number() OVER (PARTITION BY topic ORDER BY score) AS rank
        topic,
        token_1,
        token_2,
        score AS bigram_score
    FROM candidate_bigrams(50) NATURAL JOIN bigram_scores
    WHERE rank <= 5
```

Example 11 For the DBLP dataset, the result for `topic_0` is as follows.

```
 rank | topic | token_1 | token_2 | bigram_score
------+-------+---------+---------+--------------
    1 |     0 |    imag | segment |     2.336434
    2 |     0 |   model | analysi |     2.322504
    3 |     0 |    imag | classif |     2.298761
    4 |     0 |   learn |   model |     2.269486
    5 |     0 |  featur |  select |     2.260160
```

This is a vast improvement over the single token semantic label shown in Example 9. Simple string concatenation of `token_1` and `token_2` over the top 5 token bigrams gives the following semantic label for topic 0:

imag segment, model analysi, imag classif, learn model, featur select

Unfortunately, the phrases are still made of tokens rather than words. Beyond the vocabulary issues, token phrases still have a certain level of ambiguity. We do not know imag segment if it corresponds to image segmentation or image segments. While these two-word phrases are closely related, they are not entirely semantically equivalent. ■

Now, to tackle the second problem in semantic label generation. Namely, to generate words as opposed to tokens. Again, the approach in this thesis is to utilize both text analysis and the data analytical tools offered by SQL.

First, to build a mapping from words to tokens.

```
CREATE VIEW word_token(word TEXT, token TEXT) AS
    WITH W(word) AS (
        SELECT unnest(lex(text)) AS word
        FROM doc_text
    )
    SELECT word, stem(word) AS token
    FROM W
```

This produces a mapping between words and tokens, mapping each token bigram to one or more word bigrams. Now to rank the word bigrams by their document frequency. At this stage, there is no longer a need to be concerned with normalizing by the inverse document frequency of single words because it has already been done at the token level.

```
SELECT DISTINCT
    topic,
    W1.word || ' ' || W2.word AS label
    COUNT(DISTINCT doc_id) OVER (PARTITION BY (W1.word, W2.word)) AS freq,
FROM topic_token_bigrams S
    JOIN word_token W1 ON S.token_1 = W1.word
    JOIN word_token W2 ON S.token_2 = W2.word
    JOIN doc_text ON to_tsvector(title) @@ to_tsquery(W1.word || '<->' || W2.word)
```

Example 12 For the DBLP dataset, the final labels generated are shown as follows. They are from the original text and are semantically meaningful.

```
label                | freq  |rank
---------------------+-------+-----
feature selection    | 4761  |  1
image segmentation   | 4077  |  2
feature extraction   | 3577  |  3
object detection     | 3058  |  4
image classification | 2680  |  5
```

■

4.3 Using Pretrained Fasttext model

There are many aspects of the DBLP dataset that are interesting to explore. A number of them are already naturally suited to the declarative style of SQL. Exploring authors' co-authorship networks or discovering communities based on author interaction [4, 22] has been covered in the literature.

This section will explore the relationship of authors and conferences present in the DBLP dataset through the lens of the topics they cover. For this purpose, fastText is used to compute feature vectors for each title in the dataset. The fastText model used is a general purpose embedding model that was trained on Wikipedia 2017, UMBC webbase corpus and statmt.org news dataset. This model is made freely available on the fastText website [28]. It covers 16 billion tokens and has 300-dimensional vectors.

Due to the high dimensionality of fastText, dimensionality reduction is required to produce legible plots. Whenever a plot is produced in this section, Uniform Manifold Approximation(UMAP). [25] has been utilized to reduce the dimensions to 2.

One of the questions posed in Section 2.11 was about the change in topics covered by closely related conferences. Bellow will be explored how different conferences

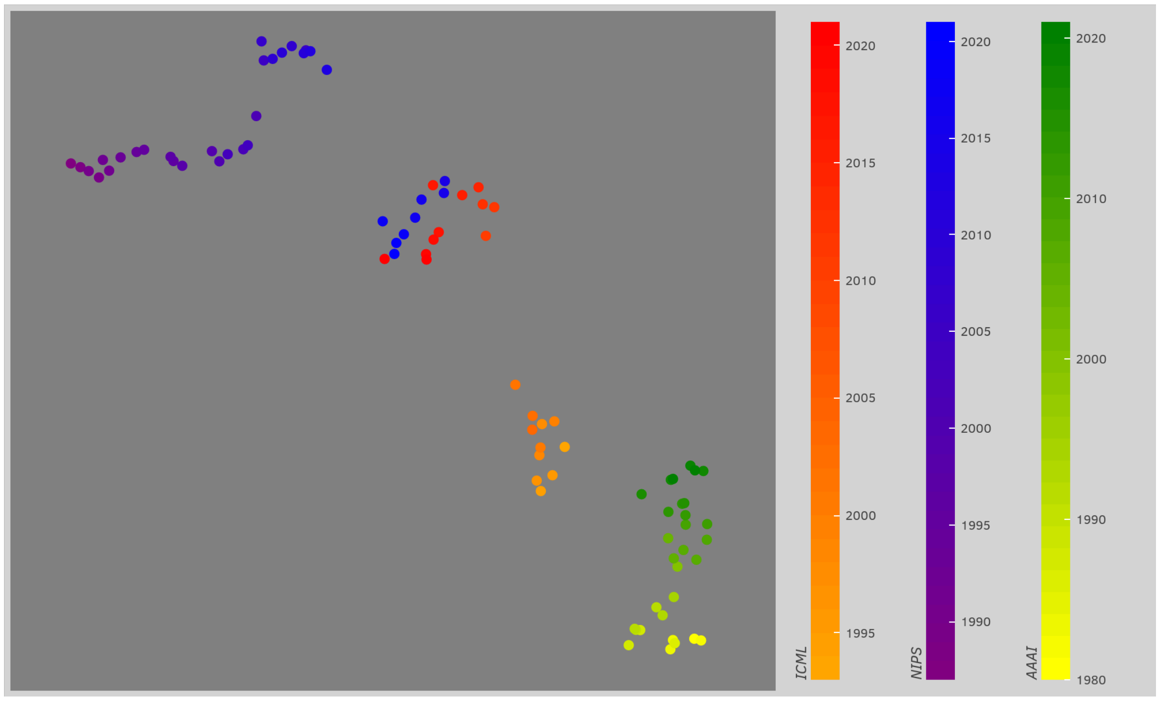


Figure 4.1: Yearly change in ML conference topics

are changing the topics they have covered over the years. Conference on Neural Information Processing Systems(NeurIPS/NIPS), International Conference on Machine Learning(ICML) and AAAI Conference on Artificial Intelligence are three distinct conferences that cover AI/Machine Learning.

To retrieve yearly vectors for each conference:

```
SELECT booktitle, year, arr_mean(feature) as f
FROM inproceedings
JOIN title_fasttext USING (id)
WHERE booktitle in ('NIPS', 'ICML', 'NeurIPS', 'AAAI')
GROUP BY booktitle, year
```

This returns:

```
 booktitle | year |   f
-----------+------+------
 AAAI      | 1980 | {...}
 AAAI      | 1982 | {...}
 AAAI       | 1983 | {...}
 AAAI       | 1984 | {...}
 AAAI       | 1986 | {...}
```

This can be visualized in Figure 4.1. it is possible to see that all 3 conferences begin covering different topics, and over time ICML and NeurIPS converge around the same point. In contrast, AAAI is covering more distinct topics even in more recent years.

AAAI has moved slowly and consistently over the years. In contrast to that both ICML and NeurIPS have taken a big some big jumps in more recent years.

We can also take a look at all of the papers covered by a small selection of top conferences from Computer Science:

- AAAI, Conference on Artificial Intelligence
- ACL, Meeting of the Association for Computational Linguistics
- COLING, International Conference on Computational Linguistics
- CSCW, ACM Conference on Computer-Supported Cooperative Work & Social Computing
- CoNLL, Conference on Computational Natural Language Learning
- EMNLP, Conference on Empirical Methods in Natural Language Processing
- FSE, Foundations of Software Engineering
- HRI, International Conference on Human-Robot Interaction
- ICCV, International Conference on Computer Vision
- ICLR, International Conference on Learning Representations

- ICML, International Conference on Machine Learning
- ICRA, IEEE International Conference on Robotics and Automation
- ICSE, the International Conference on Software Engineering
- IRI, IEEE International Conference on Information Reuse and Integration
- MICRO, IEEE/ACM International Symposium on Microarchitecture
- NIPS, Conference on Neural Information Processing Systems

In Figure 4.2, there is a distinct grouping of conferences and topics. At the same time, there is clear evidence of them overlapping in certain areas. The natural intuition is that there would be an overlap on topics such as neural networks as it is applied to a broad range of subdomains.

What about the ability to see which of our colleagues are the most similar? For this purpose, it is possible to do a nearest neighbour query and return the most similar author to each of member of the CS faculty at Ontario Tech University.

```
WITH P AS(
    SELECT author,
           arr_mean(feature) as feature,
           ROW_NUMBER() OVER (ORDER BY author) as row_id
    FROM papers_by_author(ARRAY ['Ken Q. Pu', ..., 'Heidar Davoudi'])
    JOIN title_fasttext USING (id)
    GROUP BY author
), S AS(
    SELECT a.author as a_author, b.author as b_author,
           arr_cos_sim(a.feature,b.feature) as sim
    FROM P a
    CROSS JOIN P b
    WHERE NOT a.row_id >= b.row_id
), R AS(
    SELECT *,
           ROW_NUMBER() OVER (PARTITION BY a_author ORDER BY sim DESC) as rank
```

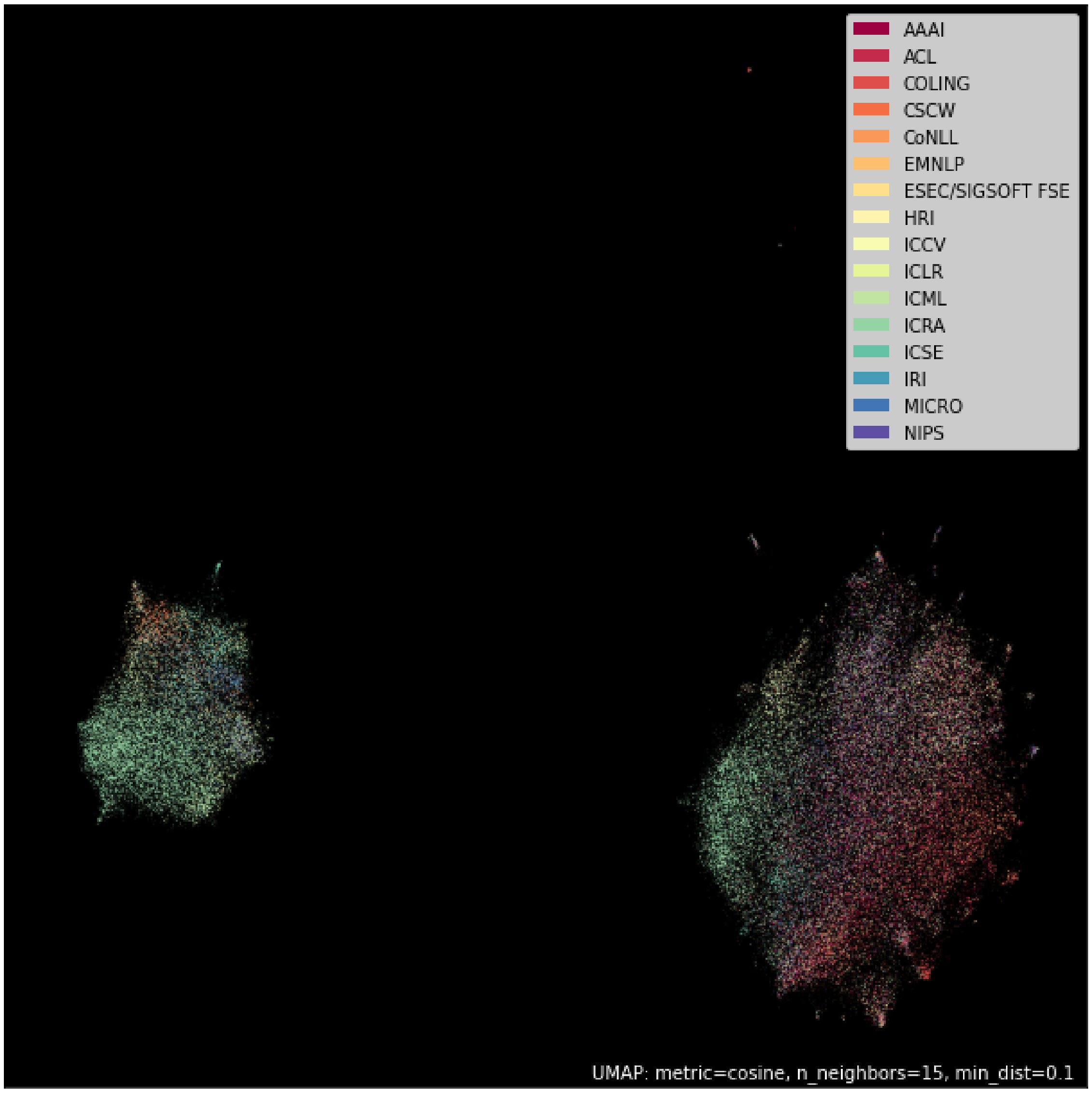


Figure 4.2: Papers from a selection of top conferences

```
    FROM S
)
SELECT * FROM R
WHERE rank = 1
ORDER BY sim DESC;
```

Here are the most similar authors in the CS faculty, with no duplicate pairs:

```
      a_author        |     b_author       |        sim         | rank
---------------------+--------------------+--------------------+------
 Jaroslaw Szlichta    | Ken Q. Pu          | 0.9604596762576774 |    1
 Heidar Davoudi       | Jaroslaw Szlichta  | 0.9366565282859791 |    1
 Faisal Z. Qureshi    | Ken Q. Pu          | 0.9348649766285365 |    1
 Jeremy S. Bradbury   | Mark Green         | 0.9292525774514236 |    1
 Ken Q. Pu            | Mark Green         | 0.8894362041671097 |    1
 Mariana Shimabukuro  | Mark Green         | 0.7518915597361522 |    1
 Christopher Collins  | Faisal Z. Qureshi  |  0.453599889779158 |    1
```

It is also possible to plot the vectors of each paper produced by these authors to gain more insight into the results. This is presented in Figure 4.3. There are certain members of faculty with much overlap and some that do not have much overlap. For example, Mark Green is notably on the edge of each cluster. While Chris Collins' papers do not overlap with other authors even though they are more central to the cluster.

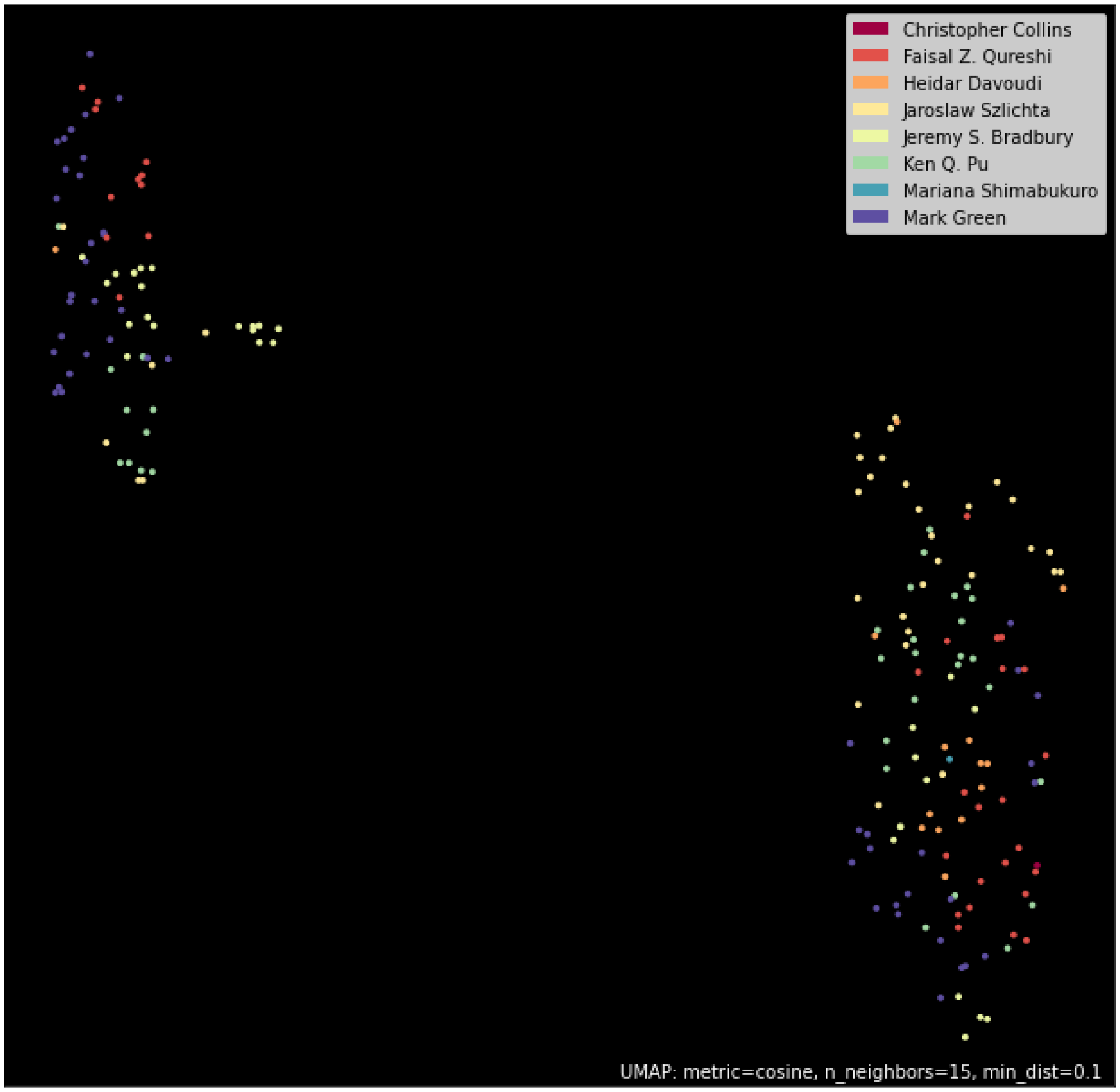


Figure 4.3: Papers of Ontario Tech CS faculty

Chapter 5

Evaluation

5.1 Qualitative Example

From the case study above, it is clear that the sorts of queries posed earlier in this thesis are very well suited for the declarative style of programming. We can see that they translate to short queries.

Aggregated vectors for yearly papers can be produced without the need for loops. This query can be modified to produce vectors for a conference or journal as a whole or over a decade or even applied to institutions and faculty. SQL provides a convenient interface to produce such vectors.

To contrast the simple query for aggregate vectors, the nearest neighbour query is more involved. Here intermediate tables are created for partial results. Even in this situation, PostgreSQL is an ally. There are methods built into the RDBMS over the years to aid in situations such as these.

All three tables created in the WITH clause can be formulated as views. This can be effective if this sort of analysis is going to take place often since any following query will not require the with clause and thus will be more straightforward. If they are materialized views, aggregate vectors must only be computed once and reused over multiple queries.

These sorts of optimizations have been built into RDBMS software over the years. With the system laid out in this thesis, it is possible to leverage them while also taking advantage of a declarative language to easily compose complex queries to answer questions about the text in large datasets such as DBLP.

5.2 Performance

To evaluate the performance of this system, different queries were run. Each query was designed in SQL and translated into Python using a Pandas DataFrame as the data storage system. This section will show each query alongside its performance result. Each query will be displayed in its SQL form as well as the translated Python/Pandas form.

To illustrate scalability, one component of each query was varied. For example, in Figure 5.1 we varied the keywords of our text query. Each result will explain its respective query and the variable, which will be varied. The time result for each step is averaged over ten runs.

In Figure 5.1 the run times of a basic query are compared. This query searches for titles of papers with specific keywords. It compares the effect of increasing the number of keywords. Adding more possible keywords to match increases the number of results returned by our queries. In Figure 5.1a we do not include the vectors in our returned results while in Figure 5.1b we also return our vector representations.

The basic SQL implementation is labelled as sql in our plot. In the case of Figure 5.1a following query was run:

```
SELECT *
FROM inproceedings
WHERE lower(title) LIKE %s
```

When the vectors are required as well, the queries are modified to perform a join with the view that contains the vectors as follows:

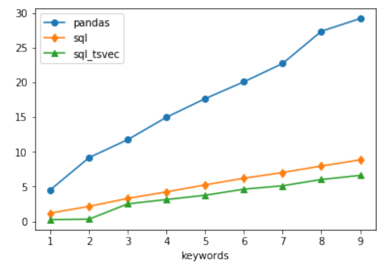
```
SELECT *
FROM inproceedings
INNER JOIN topic_vectors
    ON inproceedings.id = topic_vectors.id
WHERE lower(title) LIKE %s
```

In Figure 5.1 we also compare the basic SQL implementation where a like clause is used. With a similar query that takes advantage of PostgreSQL's full-text search. To do so, the text search vectors are computed ahead of time and are stored in a separate table. For retrieval using this system, an extra join with that table is required, and the text needs to be converted into a compatible query as follows:

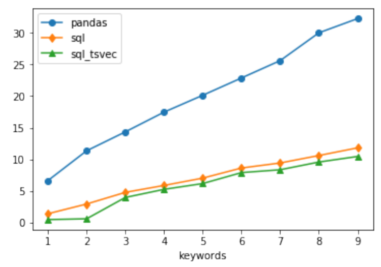
```
SELECT *
FROM inproceedings
INNER JOIN fulltext
    ON inproceedings.id = fulltext.id
WHERE fulltext.title @@ to_tsquery(%s)
```

For the Pandas implementation of this simple query, regex was used to match titles that include our keywords:

```
inproceedings[inproceedings.title.str.contains('|'.join(words),
                                               flags=re.IGNORECASE,
                                               regex=True)]
```



(a) Queries without Vectors



(b) Queries with Vectors

Figure 5.1: Text queries with and without Vectors

With these most straightforward text-based queries from Figure 5.1, it is clear that this implementation outperforms the pandas implementation significantly. Moreover, the gap widens when more keywords are used and even when the query grows to contain an unreasonable amount of keywords. While using PostgreSQL full-text search gains the SQL implementation a small margin when retrieving the vectors this gap shrinks as the additional join operation slows performance.

PostgreSQL has many features that can improve the performance of these queries. Additional care must be taken when forming the database queries and schema. In the instance above, even though PostgreSQL full-text search can perform the search over the titles column quicker than regex, it is still limited by the speed at which a join operation can be performed. While this cost is not significant when the total number of rows is small, it can be significant when there are more results. When comparing Figures 5.1a and 5.1b Most of the performance gained by using full-text search instead of LIKE is lost when the join operation is performed with a large number of rows.

Figure 5.2 compares the performance between the Pandas implementation and the PostgreSQL implementation when computing vector aggregation to produce a single vector per author. The top-n authors are ranked by the number of publications, and the list of authors is computed ahead of time in this test. However, both Pandas and PostgreSQL have methods to perform such a query in conjunction with the current

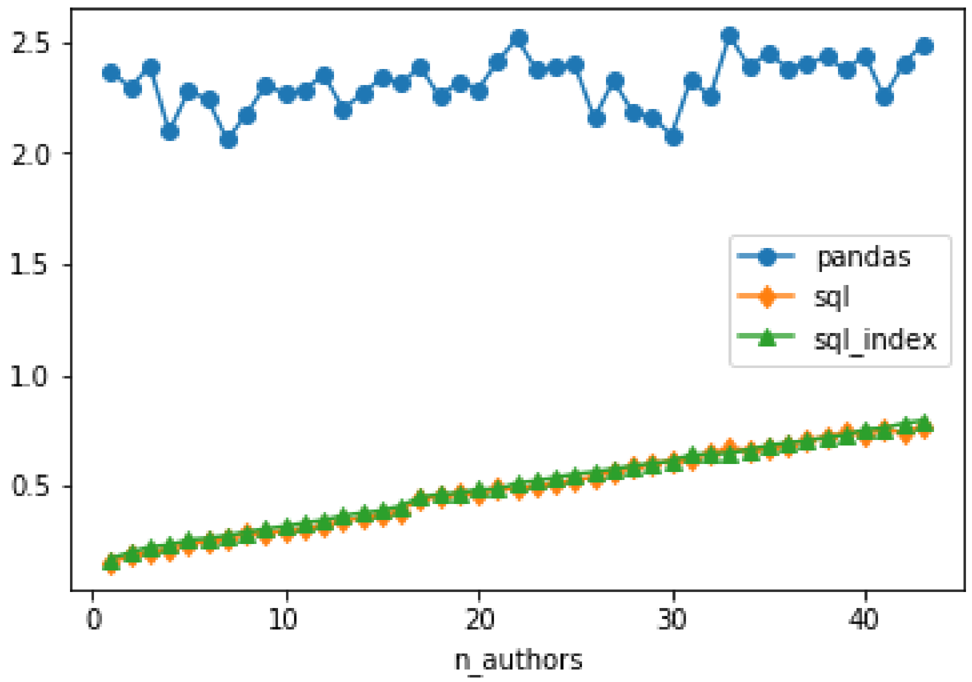


Figure 5.2: Topic vector aggregation over ton-n authors ranked by number of publications

query. Next, each author's vectors will be summated to produce the aggregated topic vector for each author.

```
docs = inproceedings[inproceedings['author'].isin(authors[:a])]
doc_vecs = pd.merge(left=docs, right=vectors, left_on='id', right_on='id')
doc_vecs.groupby(['author']).sum()
```

The Pandas implementation first selects all papers for authors in the list. Then performs a merge with the DataFrame containing our vectors. Finally, a GROUP BY and a summation compute the aggregated vectors.

```
SELECT author, count(*) as num_docs,
    sum(topic_0) as t0, sum(topic_1) as t1, sum(topic_2) as t2,
    sum(topic_3) as t3, sum(topic_4) as t4, sum(topic_5) as t5,
    sum(topic_6) as t6, sum(topic_7) as t7, sum(topic_8) as t8,
    sum(topic_9) as t9, sum(topic_10) as t10, sum(topic_11) as t11,
    sum(topic_12) as t12, sum(topic_13) as t13, sum(topic_14) as t14,
    sum(topic_15) as t15, sum(topic_16) as t16, sum(topic_17) as t17,
    sum(topic_18) as t18, sum(topic_19) as t19
FROM inproceedings
INNER JOIN topic_vectors
    ON inproceedings.id = topic_vectors.id
WHERE author in %s
GROUP BY author
```

The PostgreSQL implementation performs the same steps. Selecting by checking that the author name is in the list, then merging with the topic vector table and finally grouping by author and performing summation to produce the final author vectors. These queries were selected to be as similar in both implementations. Additionally, we ran the SQL query on a version of the database without any indexes and one with indexes to improve performance.

In the case of this last query, our indexes are not involved in the query. The titles of our papers are indexed, but that is not utilized in our query. If we where to utilize full-text search to search for papers before performing our aggregation, it would take advantage of that index to see a small speed up similar to that seen in the queries from Figure 5.1.

Due to the indexes not being utilized, the performance seen from SQL and sql_index is almost identical. Pandas performs our aggregation on multiple threads and thus does not receive a performance penalty when increasing the number of authors. The first author includes the largest number of publications, with 176, then dropping off significantly to 115 for the next author and the rest being all under 100 publications. However, even when looking at the top Forty authors, the SQL query still returns in less time than the Pandas query.

Chapter 6

Related Work

There are several fields connected to this work. This system relies on existing methods from Natural Language Processing and topic modelling, to be exact. However, the basics of this work could be extended for a number of tasks from that field. This scope will be limited to topic modelling and word embedding due to how close it is to our implementation. The topic modelling section will focus on LDA because that was the topic modelling technique that was directly implemented.

Keyword search has been a staple form of retrieving useful information from text data. We already discussed the full-text search feature built into PostgreSQL. The area of keyword search is an evolving field of research, and there are many ways to optimize the process of searching a relational database via keywords.

Qi Su et al. utilizes a custom index over the text entries within a tuple to facilitate efficient keyword-based search of records in a relational database [36]. The system proposed by the authors is an alternative interface for finding relevant records without the need to write full SQL queries. However, this search is limited to only the text columns, and tuples must first be indexed before a search is possible. The indexing process needs to happen offline before it can be used for searching. The authors also performed a user study that demonstrated the capability of keyword-based search to serve as a more searchable interface to access the text fields of an RDBMS better.

Search-as-you-type is a staple feature of web search engines. Systems such as Google's or Netflix allow the user to type and then provide results as the user is still in the process of typing their desired search. For example, Google will provide the user with suggested searches based on the text they have already typed, and Netflix will show titles that partially match what is already typed. Li et al. facilitate type-as-you-go search with pure SQL. By taking advantage of existing features to implement prefix matching, fuzzy matching [24]. They also take advantage of auxiliary tables to achieve performance suitable for an interactive search environment.

Keyword search functionality can be implemented in several ways. One common way is to build a middleware layer. Qin et al. proposed an implementation of m-keyword search queries in pure SQL [38]. An implementation such as this makes the need for writing another software layer unnecessary and cleans up the software stack in production systems. The authors have implemented three semantics of m-keyword queries: connect-tree semantics, distinct core semantics and distinct root semantics.

Similar to this work, moving NLP tasks into the RDBMS would save time and clean up the software stack. As well as improve data exploration in a framework native to relational data by adding another layer to what insight can be gathered without leaving a relational interface such as SQL.

Another method for this functionality is to generate SQL queries on demand. A framework that acts as a translation layer between human typed keyword queries and the SQL server can serve to add accessibility to data stored in the database with the need for end user knowledge of the relational structure or tables inside of the database. It can allow for data exploration without the need to create any extra indexes or a-priory access to the RDBMS [2]. The system proposed by Bergamaschi et al. takes into account the semantics of keyword queries when constructing the SQL for each query.

SODA is another system for translating keyword queries into native SQL. This system uses graph pattern matching algorithms and knowledge of the metadata model of a data warehouse [7]. This bridges the gap between knowledge stored in RDBMSs and business analysts' technical skills. Using such systems can allow business team members to more readily explore complex data with less technical knowledge of the system. For example, there is no need to familiarize a new team member with the schema before they can explore the data itself. Bridging these gaps can also speed up the rate at which new team members can get up to full speed with the work that needs to be done and the data to which they have access.

Another way this gap can be bridged is through a relaxation of the query language. Schema-free SQL [23] is a system that allows the user to under specify an SQL query in such a way that can be interpreted and a valid query can be generated. This can allow a user with no information about the underlying scheme to access data from a structured database. The system is built up of 4 parts. First, the vague query is parsed, schema elements are represented by a set of relation trees and join-paths are represented by views. Next, relation trees are matched to candidate relations in the database based on their similarity. Then, it builds a network of relations that contain all relation trees. With all information gathered through these steps, it can compose a valid SQL query to answer the original query. It can also construct nested SQL queries by processing one block at a time from the outside in.

Similar to the previous system, QUEST [3] supports unstructured queries on structured DBMSs. However, QUEST first uses Hidden Markov Models to match keywords to potential elements of the database. Next, the system combines the previously identified database elements to form join paths. These paths do not match the exact semantics of the original user query due to the vagueness involved in keyword queries. Lastly, each keyword mapping and join-paths need to be evaluated to decide which best matches the query's unstructured semantics. This system uses a probabilistic framework for scoring possible candidate queries and allows for user feedback in the form of confidence scores that are taken into account when choosing the best candidate.

These systems bridge a gap between allowing a less technical user or someone unfamiliar with the exact schema of a particular database to explore the data itself. This is complementary work to TM/SQL, which seeks to bridge the gap for people that are more technical and allow for more understanding of the text itself. When applied in conjunction with systems such as QUEST, it could be a valuable tool for data analytics teams to explore text in an easy-to-use manner that is friendly to less technical members.

LDA [6] is still a powerful topic analysis tool, and it has been able to be applied widely with great success. It is a generative model that models the relationship between a vocabulary to a number of topics. These topics are learned in an unsupervised manner, and they do not necessarily correlate directly to a topic as we can think of them, such as operating systems. Since it models the relationship between vocabulary and these latent topics, each topic is represented by a combination of the tokens that make up the vocabulary. There have been several points of improvement that have had work done to address the limitations of LDA and adapt it to better fit specific use cases.

On the internet, we often see many applications where the user is interfacing with short documents that are part of a stream. Micro-blogs, social media posts and comment sections on video platforms are all services that naturally produce short documents due to the way they are used and their limitation. These articles or posts have a temporal element that is not modelled in standard LDA. A person's tweets, for example, will change over time. As the world around them changes or they explore new areas of interest, what people discuss online will change. The authors of TM-LDA [49] address this by introducing a temporal element that tracks a transition parameter between topics for a given stream. Each account on a platform such as Twitter would have a matrix of transition parameters that can be used to predict the topic distribution of their following tweet.

Another pain point with LDA when it comes to tracking evolving topics comes when new documents are to be incorporated as they are being produced. There is no mechanism for adding new documents at a future time, and the entire model needs to be recomputed. Because of how the model allocates these latent topics, there is no straightforward way to map one topic from the model computed previously to a new model computed when new documents are created. On-line LDA [1] is an LDA implementation that allows for the model's incremental building. It allows for efficient recomputation of the topic distribution when a new document or a set of documents is presented. The priors for timestep **t-1** are used when computing the model at timestep **t**. This means that topics are carried over into the model for the next timestep. Additionally, this change in priors allows for the tracking and identification of interesting trends in topic changes.

One problem with latent topics that was hinted at earlier is that these topics do not represent concrete topics as we would describe in everyday language. For example, in a 20-topic LDA model created on a corpus of documents about computer science, no one topic clearly covers operating systems, networking or machine learning. While some of those topics can be represented, there is no guarantee that anyone is represented or that one of the topics discovered only covers one topic as people typically think of them. It could cover many topics if they are discussed in the same set of documents.

One method to tune the coherency of latent topics is by tuning and selecting the hyperparameters that best correspond to the dataset to be represented. This can be difficult and require the recomputation of the model. Hu et al. propose a method to allow interactive edits on the correlation between tokens in the corpus. [17] This method takes advantage of tree structure priors to allow for an efficient interface to the model. The user suggests their edits, the model is then updated, and the user can validate that the changes to the model are positive. The system is responsive enough that this process can happen interactively, and the user can iterate towards more coherent topics over multiple rounds.

In certain datasets, such as DBLP or Twitter, the authors of documents can be organized in networks based on their interconnection. In the DBLP dataset, relationships can be represented by co-authorship as academics write papers together on topics of common interest and in Twitter data, user relationship is modelled based on whom they follow and who follows them. iTopicModel [45] is an implementation of LDA that discards the assumption that all dataset documents are independent in favour of basing these relationships on the relationship between authors. The system considers the dependency relationships of documents based on the network structure. This change is based on the intuition that a document's topic distribution should be similar to that of its neighbours.

One method that often accompanies topic modelling is document clustering. Often applied as a secondary processing step to extract extra information about document relationships and identify closely related documents. This is possible because we can represent each topic in a vector space once topic modelling is done. The vector space is defined by the topic distribution of each document. Xie et al. present a method for performing both clustering and topic modelling in one step. [51] This system models a global distribution of topics as well as local topic distributions inside each cluster. A document has a global topic distribution, belongs to a cluster and a topic distribution local to the cluster. The authors show that clustering can help identify more coherent topics, and their system can also identify group-specific and group-independent topics.

LDA is limited by the strong independent assumption of the Dirichlet. Correlated topic model(CTM) uses a similar generative process; however, it draws its latent topic proportions from a logistic normal. [5] Whereas LDA would draw its topic proportions from a Dirichlet. This new technique can model correlations between topics. This model makes more sense when thinking about real-world use cases of algorithms such as these. When looking at a collection of articles, they are naturally connected by topics, and often multiple topics are discussed together even though they are distinct. Of course, topics are connected, especially when you consider that some topics, such as computer science and its subdomains, can be hierarchical in nature. In the DBLP dataset, all of the papers are connected by the one topic *computer science*, and all papers talk about some subtopic of this general topic. All topics that can be modelled from this dataset share some connection. Some are more interconnected, such as ML and computer vision, and others are less connected, such as database systems and computer networking. Where LDA can model a small number of topics, usually 30 or less, CTM has been capable of modelling many more topics from the same dataset, upwards of 90.

Pachinko allocation Model(PAM) [50] Seeks to solve the same limitation of LDA as CTM. However, instead of modelling relationships between topics, this model incorporates the idea of super- and sub-topics because most topics naturally organize themselves into a hierarchical structure. Take computer science as a topic. This is an extensive, all-encompassing topic. Think of it as a super topic. Many domains fall under the umbrella of computer science; data science, operating systems, networking, and machine learning. PAM models these relationships as a multi-level hierarchy of topics, where the standard LDA model is a special case with only one level of topics. Topics generated by PAM performed better in human judgment when compared to LDA. The PAM model also outperformed LDA in classification tasks.

The other way documents are connected, is by the author. For example, an academic author will work on and write about similar or interconnected topics. The Author-topic model incorporates authorship into the LDA model. [41] It is built as an extension of LDA. This new model still represents topics as a multinomial distribution over words. However, authors are now modelled as a multinomial distribution of topics. The system accounts for documents with multiple authors by representing them as a mixture of the distributions of each author. This directly models the author's topic distribution, whereas, with a traditional LDA model, we would look at the distribution of an author's overall body of work as some aggregate of the distribution that represents their work to represent the author.

LDA and similar models operate on the assumption that each of the tokens in the vocabulary is distinct. Steps such as stemming are performed to minimalism common meaning between tokens and improve the quality of the latent topics. This is difficult to do when there are documents in multiple languages. The polyLingual topic model is designed for documents authored in multiple languages. [30] One such dataset is Wikipedia. Wikipedia contains many variants of the same page containing the same information in multiple languages. Due to the web's open nature and the need for sites to service people worldwide, many web pages are available in multiple languages. PolyLDA assumes that the same topics are discussed across each language. Utilizing a data set from the previously mentioned example of Wikipedia, the authors demonstrate that it can discover aligned topics from these documents. The authors have also demonstrated that this model can be used in machine translation tasks. It was even demonstrated that this model could correctly identify the translated document as the most similar with over 75% accuracy and in the top 5% most similar with over 90% accuracy.

Work has also been done to create vector spaces based on the syntactic properties of the text. Whereas models such as LDA treat all text as a BOW. Any co-occurrence is just as relevant when treating a document as a bag of words. This is not always true when considering how natural language is structured. It matters how two words are used in the same sentence. For example, the phrases "there was something brown on the table" and "the table is brown." In the first sentence, the word brown is more relevant to the object on the table, and in the second, it describes the table itself. In the work proposed by Pado et al., the co-occurrence of two words is weighted by the relevance they have to each other depending on the context in which they appear together [33]. The system must first break down each sentence into a hierarchical parse tree, and then we can modify the score of each co-occurrence based on the relation of each word to each other.

Another NLP technique, word embeddings, can be utilized similarly to topic modelling. They both map text into a vector space. Because they are both mappings of text into a vector space, the same techniques are applicable when comparing two pieces of text. For example, one can use cosine distance as a measure of similarity between two pieces of text. The initial work in this field focused on learning embeddings of single-word tokens, and now this work has extended to applying these techniques to learning document vectors simultaneously or even to non-text-based data.

Word2vec [27] is a groundbreaking work in this field. Most modern techniques can be traced back to this work. It introduced a representation of tokens that captures the relationships between each other. Previous work has treated words in a vocabulary as entirely distinct. By considering that each word has meaning in relation to how it is used in natural language, these relationships can be learned by observing what words are used together. Through the use of a shallow neural network, this system scans over a corpus and learns a vector representation of each token. It uses a window of n tokens and a neural network that predicts the surrounding words based on the word at the center of the context window. This system was able to capture both the semantic and syntactic relationships of words in the English language.

One limitation of word2vec is that it has no way of representing documents, which are a series of tokens, naively. Assuming that the meaning of the words in a sentence is additive, we can approximate this by summing the vector representations of the individual tokens that make up a larger text, such as a sentence or paragraph. Doc2vec seeks to address this drawback by modifying word2vec's simple architecture to include a token for each document [21]. Then that document's token is included in all contexts from within the document. The embeddings for these documents are then learned in parallel with the embeddings for the vocabulary.

This early work was accomplished using a single-layer neural network. Modern neural network based word embedding models use more complicated network architectures. As well as token-level tasks such as these, Bidirectional Encoder Representations from Transformers(BERT) takes advantage of a more complicated neural network approach to tackle many tasks [12]. Devlin et al. built a general-purpose model that understands language holistically. Their pretrained model can be fine-tuned for numerous NLP tasks, from token level tasks to sentence level tasks, paraphrasing and even question and answer tasks.

Recursive neural networks are the standard modern architecture used for the creation of language models. The embeddings of single tokens are combined using different methods to produce embeddings for complex phrases or blocks of text such as sentences and documents. Many methods for this combination process have been proposed. Qian et al. specifically present one such method that is unique in that it takes into account the semantic structure of the underlying text [37]. By incorporating the Part-of-Speech(POS) tag into the combination function, the authors are able to improve in sentiment prediction tasks. Their system also incorporates embeddings for each POS tag.

Embedding models can be used for more than just words or larger bodies of text. Seq2vec is an adaptation of multi-rank embeddings, such as doc2vec, for sequential data [20]. This system works by breaking sequential data into time blocks. Then encoding the data into a vector. The resulting embedding vectors are efficient for further analysis tasks such as clustering, classification, and forecasting. Time chunk information is retained in the first rank, and subsequent ranks contain the embedding vector. The embedding is performed differently from a solution such as doc2vec. This is due to the different nature of the data. While doc2vec uses a sliding window similar to word2vec except with the inclusion of the document id as part of the context, a more complicated neural network is used here. It takes advantage of an auto-encoder network where a set of layers known as the encoder are used to create the embedding. A section called the decoder tries to recreate the original data. Here the original data itself is used as ground truth for training. This is to optimize for maximal compression of the data with minimal loss. It performs as well. Showing minimal errors in forecasting and classification tasks while taking up half the space. This could be implemented inside of the database and used on the data intake side to reduce the overall size of the table that stores this information.

Dict2vec is another word embedding model that seeks to improve the training of embedding models by enriching the training data of a traditional skip-gram model with information retrieved from the lexical dictionary [47]. This adds information about related words based on their definition. This information is added in the form of strong and weak word pairs. A strong word pair is one where each word appears in the other's definition, and a weak pair is one where only one word appears in the other's definition. These pairs are then used in the positive sampling step to ensure that related words are closer together. They also ensure that they are unrelated when choosing negative samples. Where a negative sample is chosen randomly, there is a minimal chance that this negative sample is still related to the word. These word pairs can be used as a baseline to eliminate these related words. Ensuring that related pairs are not moved further apart. The results show significant improvements in similarity when compared to human evaluation.

AMETHYST [10] is an existing system that employs NLP to organize and help find topics in heterogeneous data. It focuses on the organization of topics and keywords into hierarchies. In contrast Itoh et al. [18] propose a system to visualize the change in topics of a blog over time. Their approach visualizes a slice of time on one plane, and a stack of these planes show the change of topics across time.

Tang et al. [46] proposed a system to compute the similarity between two domains. Using the same dataset, they track changes in similarity and co-authorship between domains. They manually identified time periods and domains for the segmentation of data. Song et al. [43] use clustering to identify trends in bioinformatics field. Using the same dataset, they partition it into four non-overlapping periods and use Markov Random Field-based topic clustering for topic extraction. They analyze both within-period and between-period cluster similarity. Doumit et al. [13] apply LDA along with NLP to extract the bias of news sources. This combination of LDA and NLP shows that it is possible to discern each news outlet's bias in their coverage of an event and identify the themes presented by their coverage. Mei et al. [26] take a probabilistic approach to identify the evolution of a theme over time. They use a probabilistic mixture model to extract themes and Hidden Markov Models to model the transition between themes.

Jelodar et al. [19] provide a comprehensive review of LDA topic modelling methods. Their work covers many parameter estimation methods, applications, and adaptations for specific domains. One such adaptation is to deal with short documents such as Tweets, and others address other issues such as enriching topic vectors with timestamped data.

Chapter 7

Conclusions

7.1 Summary

In this thesis, a framework for implementing NLP operators inside of an existing RDBMS was presented. Performing NLP operations alongside traditional relation data queries allows for a richer analysis of both types of data. This analysis can now be performed from a single relational interface. It empowers a more complex understanding of existing data.

Relational databases have tremendous power to store and analyze large quantities of data. This data can be a mix of text and numeric values. While numeric values are easy to compare and perform calculations with, there are no built-in tools for analyzing text in a way that allows us to compare entries. Basic features such as regex and full-text search allow some information to be extracted.

Building NLP operators, such as for topic modelling, into the relational database allows for deeper analysis of text entries. At the topic model level, we can compare the content of text, and this is because we can transform text, which is very computationally strenuous to compare, into a vector space where we can easily compute a variety of metrics that allow us to gather statistics and comparisons.

Using the power of RDBMS for NLP, a novel method of generating labels for LDA topic models was presented. This method takes advantage of standard SQL and some PostgreSQL features alongside the new topic modelling implementation to work backwards and create word phrases. Working backwards from tokens that are typically used are representations for LDA topics to words and phrases creates candidate topic labels that can better capture the semantics of the original words used in the text.

This framework applies to newer embedding techniques and those not yet discovered. This principle was exemplified by analyzing the same DBLP dataset with FastText. Naturally, it does not stop there. BERT has proven to be very useful for many NLP tasks, and this framework could easily be extended to be compatible with that. As well as possible any new text embedding techniques to come in the future

7.2 Limitations

There are several limitations when it comes to performing NLP operations inside of the RDBMS. The plpythonu module in PostgreSQL is very powerful, but that comes at a disadvantage. This implementation is only limited to PSQL. Even though other relational database systems allow for similar functionality, this work only focuses on implementation in PSQL.

PostgreSQL's pl/pythonu module adds python support as an untrusted language. This means that it does not restrict Python's features. This comes at a cost; only superusers can define new functions that use this language, which is not enabled by default. This means it must first be enabled before any functions can be defined. The result is that functions that contain Python cannot be introduced in an ad-hoc way. The implementation must be carefully planned to enable as much to be done in standard SQL.

One way to overcome this is still to perform embedding outside of the database. Then use the database engine for its analytical powers. Doing the embedding traditionally also alleviates concerns about file access through pl/pythonu on the server.

7.3 Future work

A few obvious extensions appear from this work. NLP does not only entail topic modelling. There are some modern NLP tasks that can be performed inside of the RDBMS. Not only because it is possible but because it can enrich the other tasks that are naturally performed in SQL. PostgreSQL is not the only RDBMS with features that enable this functionality.

BERT [12] is a good fit as a general-purpose ML model for numerous NLP tasks. Implementing this model would enrich the data we have been working with even further with a deeper understanding of the text data used in this work. Its rich understanding of text can be incorporated, creating vector representations of text documents and used in either classification or clustering. Moreover, this model can allow for more complicated tasks such as generating new text [48] or identifying existing answers to questions.

Other database systems have support for functions in other languages, enabling implementations of similar systems. Namely, SQLite3 and Microsoft SQL Server have the necessary Python support that enables such systems' implementation. While they are both excellent candidates, SQLite3 is limited by the fact that any python functions introduced are limited to that session and do not transfer to other clients, and it is not entirely suited for complicated tasks such as NLP.

Microsoft SQL server added support for Python and R only after this work started. Similar functionality may appear in the future. Any such database is suitable for integrating NLP tasks as long as it can support functions written in a standard language for machine learning or data science and share objects between calls to the same function. Sharing objects between function calls is essential for performance.

Bibliography

[1] AlSumait, L., Barbará, D., and Domeniconi, C. On-line LDA: Adaptive Topic Models for Mining Text Streams with Applications to Topic Detection and Tracking. In 2008 Eighth IEEE International Conference on Data Mining (Dec. 2008), pp. 3–12. ISSN: 2374-8486.

[2] Bergamaschi, S., Domnori, E., Guerra, F., Trillo Lado, R., and Velegrakis, Y. Keyword search over relational databases. Proceedings of the 2011 international conference on Management of data - SIGMOD '11 (jun 2011), 565.

[3] Bergamaschi, S., Guerra, F., Interlandi, M., Trillo-Lado, R., and Velegrakis, Y. QUEST: a keyword search system for relational data based on semantic and machine learning techniques. Proceedings of the VLDB Endowment 6, 12 (Aug. 2013), 1222–1225.

[4] Biryukov, M., and Dong, C. Analysis of computer science communities based on dblp. In International Conference on Theory and Practice of Digital Libraries (2010), Springer, pp. 228–235.

[5] Blei, D. M., and Lafferty, J. D. Correlated topic models. Advances in Neural Information Processing Systems (2005), 147–154.

[6] Blei, D. M., Ng, A. Y., and Jordan, M. I. Latent dirichlet allocation. Journal of machine Learning research 3, Jan (2003), 993–1022.

[7] Blunschi, L., Jossen, C., Kossman, D., Mori, M., and Stockinger, K. SODA: Generating SQL for Business Users. arXiv:1207.0134 [cs] (June 2012). arXiv: 1207.0134.

[8] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. Enriching Word Vectors with Subword Information. arXiv:1607.04606 [cs] (June 2017). arXiv: 1607.04606.

[9] Codd, E. F. A Relational Model of Data for Large Shared Data Banks. 11.

[10] Danilevsky, M., Wang, C., Tao, F., Nguyen, S., Chen, G., Desai, N., Wang, L., and Han, J. AMETHYST: a system for mining and exploring topical hierarchies of heterogeneous data. In Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '13 (Chicago, Illinois, USA, 2013), ACM Press, p. 1458.

[11] Dean, J., and Ghemawat, S. Mapreduce: Simplified data processing on large clusters. Commun. ACM 51, 1 (jan 2008), 107–113.

[12] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs] (May 2019). arXiv: 1810.04805.

[13] Doumit, S., and Minai, A. Online News Media Bias Analysis using an LDA-NLP Approach. 16.

[14] El-Kishky, A., Song, Y., Wang, C., Voss, C., and Han, J. Scalable Topical Phrase Mining from Text Corpora. arXiv:1406.6312 [cs] (Nov. 2014). arXiv: 1406.6312.

[15] Hotho, A., Nurnberger, A., Paass, G., and Augustin, S. A Brief Survey of Text Mining. LDV Forum 20 (2005), 19–62.

[16] Hu, D. J. Latent Dirichlet Allocation for Text, Images, and Music. University of California, San Diego 26 (2009), 19.

[17] Hu, Y., Boyd-Graber, J., Satinoff, B., and Smith, A. Interactive topic modeling. Machine Learning 95, 3 (June 2014), 423–469.

[18] Itoh, M., Yoshinaga, N., Toyoda, M., and Kitsuregawa, M. Analysis and visualization of temporal changes in bloggers' activities and interests. In 2012 IEEE Pacific Visualization Symposium (Feb. 2012), pp. 57–64. ISSN: 2165-8773.

[19] Jelodar, H., Wang, Y., Yuan, C., Feng, X., Jiang, X., Li, Y., and Zhao, L. Latent Dirichlet allocation (LDA) and topic modeling: models, applications, a survey. Multimedia Tools and Applications 78, 11 (June 2019), 15169–15211.

[20] Kim, H. J., Hong, S. E., and Cha, K. J. seq2vec: Analyzing sequential data using multi-rank embedding vectors. Electronic Commerce Research and Applications 43 (Sept. 2020), 101003.

[21] Le, Q., and Mikolov, T. Distributed representations of sentences and documents. In Proceedings of the 31st International Conference on Machine Learning (Bejing, China, 22–24 Jun 2014), E. P. Xing and T. Jebara, Eds., vol. 32 of Proceedings of Machine Learning Research, PMLR, pp. 1188–1196.

[22] Leskovec, J., and Krevl, A. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[23] Li, F., Pan, T., and Jagadish, H. V. Schema-free SQL. In Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (Snowbird Utah USA, June 2014), ACM, pp. 1051–1062.

[24] Li, G., Feng, J., and Li, C. Supporting Search-As-You-Type Using SQL in Databases. IEEE Transactions on Knowledge and Data Engineering 25, 2 (Feb. 2013), 461–475. Conference Name: IEEE Transactions on Knowledge and Data Engineering.

[25] McInnes, L., Healy, J., and Melville, J. Umap: Uniform manifold approximation and projection for dimension reduction, 2018.

[26] Mei, Q., and Zhai, C. Discovering evolutionary theme patterns from text: an exploration of temporal text mining. In Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining (Chicago, Illinois, USA, Aug. 2005), KDD '05, Association for Computing Machinery, pp. 198–207.

[27] Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient Estimation of Word Representations in Vector Space. arXiv preprint arXiv:1301.3781 (Sept. 2013). arXiv: 1301.3781.

[28] Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., and Joulin, A. Advances in pre-training distributed word representations. In Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018) (2018).

[29] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. Distributed Representations of Words and Phrases and their Compositionality. In Advances in Neural Information Processing Systems (oct 2013), vol. 60, pp. 1555–1563.

[30] Mimno, D., Wallach, H. M., Naradowsky, J., Smith, D. A., and McCallum, A. Polylingual topic models. In EMNLP 2009 - Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: A Meeting of SIGDAT, a Special Interest Group of ACL, Held in Conjunction with ACL-IJCNLP 2009 (2009), pp. 880–889.

[31] Moody, C. E. Mixing Dirichlet Topic Models and Word Embeddings to Make lda2vec. arXiv:1605.02019 [cs] (May 2016). arXiv: 1605.02019.

[32] Ordonez, C. Programming the K-Means Clustering Algorithm in SQL. In Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (New York, NY, USA, 2004), KDD '04, Association for Computing Machinery, pp. 823–828.

[33] Padó, S., and Lapata, M. Dependency-based construction of semantic space models. Computational Linguistics 33, 2 (jun 2007), 161–199.

[34] Pennington, J., Socher, R., and Manning, C. Glove: Global Vectors for Word Representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP) (Doha, Qatar, 2014), Association for Computational Linguistics, pp. 1532–1543.

[35] Pitchaimalai, S. K., Ordonez, C., and Garcia-Alvarado, C. Efficient Distance Computation Using SQL Queries and UDFs. In 2008 IEEE International Conference on Data Mining Workshops (2008), pp. 533–542.

[36] Qi Su, and Widom, J. Indexing relational database content offline for efficient keyword-based search. In _9th International Database Engineering Application Symposium (IDEAS'05)_ (July 2005), pp. 297–306. ISSN: 1098-8068.

[37] Qian, Q., Tian, B., Huang, M., Liu, Y., Zhu, X., and Zhu, X. Learning Tag Embeddings and Tag-specific Composition Functions in Recursive Neural Network. In _Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)_ (Beijing, China, 2015), Association for Computational Linguistics, pp. 1365–1374.

[38] Qin, L., Yu, J. X., and Chang, L. Keyword search in databases. _Proceedings of the 2009 ACM SIGMOD International Conference on Management of data_ (jun 2009), 681–694.

[39] Radford, A., Jozefowicz, R., and Sutskever, I. Learning to generate reviews and discovering sentiment.

[40] Řehůřek, R., and Sojka, P. Software Framework for Topic Modelling with Large Corpora. In _Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks_ (Valletta, Malta, May 2010), ELRA, pp. 45–50. `http://is.muni.cz/publication/884893/en`.

[41] Rosen-Zvi, M., Griffiths, T., Steyvers, M., and Smyth, P. The Author-Topic Model for Authors and Documents. _arXiv preprint arXiv:1207.4169_ (jul 2012).

[42] Sang, E. F. T. K., and De Meulder, F. Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. arXiv preprint cs/0306050 (jun 2003).

[43] Song, M., Heo, G. E., and Kim, S. Y. Analyzing topic evolution in bioinformatics: investigation of dynamics of the field with conference data in DBLP. Scientometrics 101, 1 (Oct. 2014), 397–428.

[44] Sridhar, K. T. Big Data Analytics Using SQL: Quo Vadis? In Research and Practical Issues of Enterprise Information Systems (Cham, 2018), A. M. Tjoa, L.-R. Zheng, Z. Zou, M. Raffai, L. D. Xu, and N. M. Novak, Eds., Springer International Publishing, pp. 143–156.

[45] Sun, Y., Han, J., Gao, J., and Yu, Y. iTopicModel: Information Network-Integrated Topic Modeling. In 2009 Ninth IEEE International Conference on Data Mining (Miami Beach, FL, USA, Dec. 2009), IEEE, pp. 493–502.

[46] Tang, X., Yang, C. C., and Song, M. Understanding the evolution of multiple scientific research domains using a content and network approach. Journal of the American Society for Information Science and Technology 64, 5 (2013), 1065–1075. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/asi.22813.

[47] Tissier, J., Gravier, C., and Habrard, A. Dict2vec : Learning word embeddings using lexical dictionaries. In EMNLP 2017 - Conference on Empirical Methods in Natural Language Processing, Proceedings (2017), Association for Computational Linguistics (ACL), pp. 254–263.

[48] Wang, A., and Cho, K. BERT has a Mouth, and It Must Speak: BERT as a Markov Random Field Language Model. arXiv preprint arXiv:1902.04094 1, 2 (feb 2019).

[49] Wang, Y., Agichtein, E., and Benzi, M. TM-LDA: efficient online modeling of latent topic transitions in social media. In Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '12 (Beijing, China, 2012), ACM Press, p. 123.

[50] Wei, L., and McCallum, A. Pachinko allocation: DAG-structured mixture models of topic correlations. In ACM International Conference Proceeding Series (New York, New York, USA, 2006), vol. 148, ACM Press, pp. 577–584.

[51] Xie, P., and Xing, E. P. Integrating document clustering and topic modeling. In Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence (Arlington, Virginia, USA, Aug. 2013), UAI'13, AUAI Press, pp. 694–703.