People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

Chadli Bendjedid University
Faculty of Sciences & Technologies
Computer Science Department

Master Thesis

Design and Implementation of an Intelligent MOOC Platform (Massive open online course)

Submitted in partial fulfillment of the requirements for a
Master's degree in Computer Science
Option: Intelligent Computer Systems

By
Sahia Abd Elhadi

IN FRONT OF THE JURY COMPOSED OF:

Quality	Name and Surname	Rank	University
President:	DR. Bentrad S	MCB	Chadli Bendjedid El-Tarf
Supervisor:	DR. BANMACHICHE A	MCA	Chadli Bendjedid El-Tarf
Examiner:	DR. Chemam C	MAA	Chadli Bendjedid El-Tarf

UNIVERSITY YEAR: 2023/2024

Acknowledgments

First and foremost, I am profoundly grateful to Allah for His guidance, wisdom, and blessings, which have illuminated my path and given me the strength to persevere and succeed. Without His divine intervention, none of this would have been possible.

I would like to extend my deepest gratitude to my parents, whose unwavering love, support, and encouragement have been the cornerstone of my journey. Their sacrifices and prayers have been the driving force behind my achievements.

To my dear little sisters, Oumaima and Noor, your innocent smiles and boundless affection have been a source of joy and motivation. Your presence in my life has been a constant reminder of the importance of perseverance and dedication.

A special note of thanks to my esteemed supervisor, Dr. Benmachich Abd Elmajid, whose expertise, guidance, and feedback have been instrumental in shaping this work. Your mentorship has been a beacon of knowledge and inspiration, and I am deeply grateful for your unwavering support.

To my friends who have contributed to the realization of this work, especially Anis, Loukman, and Salah, your assistance, collaboration, and camaraderie have been invaluable. Your dedication and friendship have made this journey a rewarding and memorable experience.

Furthermore, I extend my heartfelt thanks to all my other friends, including those unnamed individuals who have offered their support and encouragement. Your contributions, whether big or small, have not gone unnoticed and are deeply appreciated.

In conclusion, I am both humbled and profoundly grateful for the support, guidance, and contributions of all those mentioned and unmentioned in this acknowledgment. Their involvement has significantly enriched this research journey, and I am deeply indebted to each and every one of them. This work stands as a testament to the collective effort and unwavering support of a remarkable community, for which I am eternally grateful.

Sincerely, Sahia Abd Elhadi

Dedication

This work is dedicated to my parents, whose unwavering love, sacrifices, and prayers have been the foundation of my success. Your endless support has been my greatest source of strength and inspiration.

To my two precious sisters, Oumaima and Nour, your smiles and innocence have been my greatest motivation. You are my source of joy, and I hope this work inspires you to chase your dreams with all your heart.

To my esteemed supervisor, Dr. Benmachich Abd Elmajid, your guidance and mentorship have been a beacon of knowledge and inspiration. This work stands as a testament to your dedication and expertise.

To my dear friends, especially Kezihi Salah Eddine, Anis Benflis, and Loukman Touil, your camaraderie and assistance have made this journey unforgettable. Your support has been instrumental in the realization of this work.

And finally, to all those unnamed individuals whose contributions and encouragement have enriched this journey, this work is a tribute to your collective effort and unwavering support.

With heartfelt gratitude,

Sahia Abd Elhadi

Abstract

The exponential expansion of Massive Open Online Courses (MOOCs) has underscored the critical importance of efficient search mechanisms within digital learning ecosystems. This thesis presents an innovative MOOC-specific search engine, capitalizing on recent breakthroughs in natural language processing (NLP) and deep learning.

The primary objective of this research is to develop a cohesive, modular search platform tailored explicitly to the diverse requirements of MOOC environments. The modularity inherent in the system architecture ensures adaptability and scalability, facilitating the independent development, testing, and refinement of individual components.

Key components of this advanced search engine include Siamese networks emphasizing contrastive learning, Long Short-Term Memory (LSTM) networks for contextual language representation, and graph neural networks. These components bolster text processing and comprehension, thereby ensuring the delivery of accurate and relevant search results.

Recent challenges in the realm of advanced search and retrieval systems within MOOCs are systematically addressed, with proposed solutions leveraging cutting-edge NLP and machine learning technologies. The resulting systems offer high degrees of customization, enabling the provision of precise and pertinent search outcomes.

Experimental findings showcase the enhanced accuracy of the modular search engine, laying a robust groundwork for future advancements in MOOC search technologies. The system remains adaptable and versatile, poised for further exploration aimed at enriching user engagement and optimizing learning outcomes within digital educational platforms

ملخص

شهد النمو الهائل للدورات الدراسية العبر الإنترنت الضخمة أهمية حاسمة لآليات البحث الفعالة ضمن النظم التعليمية الرقمية. يقدم هذا الأطروحة محرك بحث مبتكر مخصص للدورات الدراسية العبر الإنترنت الضخمة، مستفيداً من التطورات الحديثة في مجال معالجة اللغة الطبيعية والتعلم العميق.

الهدف الرئيسي لهذا البحث هو تطوير منصة بحث متكاملة ومرنة مصممة خصيصاً لتلبية احتياجات البيئات التعليمية للدورات الدراسية العبر الإنترنت الضخمة. يضمن التقسيم الوحداتي المتميز في بنية النظام المرونة والقابلية للتوسع، مما يسهل عمليات التطوير والاختبار والتحسين المستقلة للمكونات الفردية.

تشمل المكونات الرئيسية لهذا المحرك المتقدم للبحث الشبكات المزدوجة التي تؤكد على التعلم التناقضي، وشبكات الذاكرة القصيرة الطويلة لتمثيل اللغة السياقية، وشبكات الجراف التعليمية. تعزز هذه المكونات معالجة النصوص وفهمها، مما يضمن تقديم نتائج بحث دقيقة وملائمة.

يتم معالجة التحديات الحديثة في مجال أنظمة البحث والاسترجاع المتقدمة داخل دورات الدراسية العبر الإنترنت الضخمة بشكل نظامي، مع اقتراح حلول تستفيد من التكنولوجيات الحديثة في مجال معالجة اللغة الطبيعية والتعلم الآلي. تقدم الأنظمة الناتجة درجات عالية من التخصيص، مما يتيح تقديم نتائج بحث دقيقة ومناسبة.

تعكس النتائج التجريبية تحسين دقة المحرك المتقدم للبحث، مما يضع الأسس الراسخة للتقدم في تكنولوجيا البحث لدورات الدراسية العبر الإنترنت الضخمة. يظل النظام مرنًا ومتنوعًا، جاهزًا لاستكشافات مستقبلية تهدف إلى تحسين مشاركة المستخدم وتحسين نتائج التعلم داخل المنصات التعليمية الرقمية.

Résumé

L'expansion exponentielle des Massive Open Online Courses (MOOCs) a souligné l'importance critique des mécanismes de recherche efficaces au sein des écosystèmes d'apprentissage numériques. Cette thèse présente un moteur de recherche spécifique aux MOOCs, capitalisant sur les avancées récentes en traitement automatique du langage naturel (TALN) et en apprentissage profond.

L'objectif principal de cette recherche est de développer une plateforme de recherche cohésive et modulaire, spécifiquement adaptée aux besoins diversifiés des environnements MOOC. La modularité inhérente à l'architecture du système garantit l'adaptabilité et la scalabilité, facilitant le développement, le test et le raffinement indépendants des composants individuels.

Les composants clés de ce moteur de recherche avancé comprennent des réseaux siamois mettant l'accent sur l'apprentissage contrastif, des réseaux à mémoire à long terme et à court terme (LSTM) pour la représentation contextuelle du langage, et des réseaux neuronaux graphiques. Ces composants renforcent le traitement et la compréhension des textes, assurant ainsi la livraison de résultats de recherche précis et pertinents.

Les défis récents dans le domaine des systèmes avancés de recherche et de récupération au sein des MOOCs sont systématiquement abordés, avec des solutions proposées qui exploitent les technologies de pointe en TALN et en apprentissage automatique. Les systèmes résultants offrent des degrés élevés de personnalisation, permettant la fourniture de résultats de recherche précis et pertinents.

Les résultats expérimentaux mettent en avant l'amélioration de la précision du moteur de recherche modulaire, jetant des bases solides pour les avancées futures dans les technologies de recherche des MOOCs. Le système reste adaptable et polyvalent, prêt pour une exploration plus approfondie visant à enrichir l'engagement des utilisateurs et à optimiser les résultats d'apprentissage au sein des plateformes éducatives numériques.

Contents

List of Figures

List of Tables

General Introduction

The advancement of Internet technology, particularly wireless Internet, has significantly enhanced online education. E-learning has become a crucial educational approach, especially during the COVID-19 pandemic, and is expected to remain a popular alternative to traditional teaching methods. Massive Open Online Courses (MOOCs) represent a cutting-edge form of e-learning that has transformed the educational landscape.

Introduced in 2008 by Stephen Downes and George Siemens, MOOCs provide access to a wide range of topics and educational materials to learners worldwide, regardless of age, gender, race, or geographic location. Platforms like Coursera, edX, and FutureLearn offer free courses from prestigious universities, making high-quality education accessible to millions.

MOOCs offer significant flexibility in terms of time and location, and promote interaction through discussion forums, enhancing student engagement and knowledge sharing. They shift the role of educators from sole knowledge providers to mentors, focusing on the individual needs of learners. By empowering students to rely on themselves, MOOCs help educators act as guides and counselors rather than the exclusive source of knowledge.

To mitigate these challenges, various strategies have been developed, including enhancing the quality of course materials, integrating interactive elements, and providing support systems for students. MOOCs leverage advanced technologies to offer personalized learning experiences, foster engagement through virtual communities, and provide continuous feedback to learners. These efforts aim to improve user satisfaction, retention rates, and overall learning outcomes.

This introduction sets the stage for examining the pivotal role of MOOCs in modern education. By understanding their mechanisms, challenges, and applications, we can appreciate the opportunities and complexities that MOOCs present. Through further exploration, we can gain deeper insights into how MOOCs are reshaping the educational landscape and democratizing access to knowledge.

The organization of this thesis is given as follows:

- **State of the Art**

  In this chapter, we provide an extensive overview of the current landscape of Massive Open Online Courses (MOOCs). We explore the foundational concepts of MOOCs, their evolution, and their significance in modern education. Additionally, we examine the various pedagogical approaches, content delivery methods, and learner engagement techniques employed within MOOC platforms. Furthermore, we discuss emerging trends in machine learning and AI that are influencing the development of MOOCs, such as personalized learning algorithms and recommendation systems.

- **Conceptual Desig**

  The second chapter focuses on the conceptual design of the components to be implemented to enhance MOOC platforms. We outline the specific functionalities and technologies we intend to incorporate, such as personalized learning algorithms, content recommendation systems, and natural language processing for content understanding. Additionally, we discuss the architectural design of these components and the methodologies for dataset selection, model training, and evaluation.

- **Results and Implementation**

  In the final chapter, we present the implementation results and analysis of the proposed components for enhancing MOOC platforms. We provide insights into the representation of the development tools and frameworks utilized in building these components. Experimental results are analyzed to showcase their effectiveness and efficiency in addressing existing challenges within MOOCs. Additionally, we discuss the implications of our findings and potential avenues for future research in the field.

Chapter 1

State of the Art

1.1 Introduction

Massive Open Online Courses (MOOCs) have redefined access to education, providing a platform for learners worldwide to engage with diverse content. MOOCs utilize digital technologies to democratize learning, transcending geographical and socio-economic barriers.

In this chapter, we delve into the essence of MOOCs and the technological innovations driving their evolution. We explore the interactive nature of MOOC platforms, where learners engage with course materials through various mediums, fostering a dynamic learning environment.

1.2 Massive open online courses(MOOC)

Massive Open Online Courses (MOOCs) are accessible online courses available to anyone, offering a cost-effective and flexible means to acquire new skills and advance careers while providing high-quality education on a large scale.

Millions globally turn to MOOCs for various purposes, such as career advancement, career transitions, college preparation, supplementary education, lifelong learning, and corporate training.

MOOCs have revolutionized global education, significantly altering traditional learning paradigms.

1.3 Approaches in MOOC Platforms

Various approaches have been developed to address challenges within MOOC platforms. These approaches leverage different technologies and methodologies to enhance the learning experience and improve educational outcomes.

1.3.1 Simple Approaches

- **Reinforcement Learning for Personalized Suggestions:** This approach uses reinforcement learning theory to generate individualized suggestions through user assessment metrics, enhancing personalized recommendations by replacing traditional similarity calculations with parameter approximation based on user state value functions.
- **Attentional Heterogeneous Graph Convolutional Deep Knowledge Recommender (ACKRec):** Builds a heterogeneous information network (HIN) including courses, videos, and pro-

fessors to identify semantic links between different entities. Utilizes attention mechanisms to spread user preferences in a meta-path-directed manner, using data from the XuetangX MOOC platform.

- **Semantic-Based Recommendation System:** Utilizes past information to match users' needs, offering learners individualized suggestions based on interests, talents, and preferences, thereby improving engagement and learning experiences.
- **Multi-Theoretical Mixed Model:** Combines a smart learning environment with a technological acceptance method and a learning satisfaction model, using Partial Least Squares Structural Equation Modeling (PLS-SEM) to analyze factors influencing user behavior, including attitudes, perceived utility, and usability.
- **Real-Time Student Engagement Tracking:** Applies a CNN algorithm to survey data collected during lectures to track student involvement. Helps academic staff understand factors affecting student engagement and enables teachers to evaluate performance based on engagement ratings.
- **Hierarchical Text Classification:** Uses deep learning to combine words into vectors and messages into final vectors, identifying important words and phrases within the emotional context to enhance classification accuracy, using the Stanford MOOCPosts dataset.
- **Sentiment Analysis for Personalized MOOCs:** Uses natural language processing libraries to understand student preferences, allowing for the creation of personalized MOOCs. Analyzes reviews to cluster MOOC titles and sentiments related to courses and instructors.
- **Graph Convolutional Network-Based Approach:** Represents data on MOOC platforms as a Heterogeneous Information Network (HIN). Builds user and concept representations from HIN's meta-paths to predict concept preference evaluations for recommendations, using the MOOC Cube dataset from XuetangX.
- **Deep Neural Networks for Mixed-Mode Learning:** Evaluates online and offline mixed-mode learning impacts using semi-wrapped Gaussian distribution and dual parallel training to assess MOOC evaluations and improve recommendation accuracy.
- **Deep Neural Network for Satisfaction Scores:** Predicts satisfaction scores by analyzing student learning behaviors, converting data from activities like video-watching and exercise completion into metrics to enhance model accuracy, using the National Tsing Hua University (NTHU) dataset.
- **Behavior Tree-Based Curriculum Recommendation:** Integrates techniques like TextCNN and BiLSTM to boost text input usage and extract reliable data. Uses the Canvas network dataset and the MOOC dataset from Chinese institutions to improve course suggestion accuracy.
- **Reinforcement Learning in MOOC Recommendations:** Suggests practice activities based on learners' current comprehension levels using the deep deterministic policy gradient algorithm (DDPG). Refines recommendations through feedback on exercise performance, using data from the NTHU MOOCs platform.
- **Fractional-Iterative Deep BiLSTM (F-itr Deep BiLSTM):** Predicts student dropout by analyzing knowledge data using statistical and machine-learning techniques. Handles large datasets effectively, using the KDD Cup 2015 data on MOOC dropout prediction.

1.3.2 Hybrid Approaches

- **Combining NLP, CNN, and RNN Models:** Enhances learning management systems by mining association rules to determine time allocation and using a semantic method to combine cluster and association rule-finding scores.
- **Combining CNNs with Feature Weighting and Time Series Analysis (FWTS-CNN):** Predicts student dropout rates by evaluating clickstream data and learning behaviors, showing improved prediction accuracy for large-data MOOCs.
- **GRU and RNN Models for Student Progress Prediction:** Integrates static and sequential data to identify at-risk students in online courses. Uses data completion techniques to fill gaps, analyzed with the Open University Learning Analytics Dataset (OULAD).
- **Decision Trees and Artificial Neural Networks for Academic Achievement Prediction:** Assesses factors influencing student success and uses multi-optimizers to enhance prediction accuracy, trained on a customized learning dataset from system databases.
- **CNN and RNN Integration for Dropout Prediction:** Uses CNN techniques to extract features, integrated with Recurrent Neural Networks (RNN) for dropout prediction. Refines features from clickstream log data and inputs them into a logistic regression classification model.

1.4 Search Engines

Overview

Search engines are designed to search for information on the World Wide Web and return relevant results based on user queries. They index web pages, images, videos, and other types of content, making it easier for users to find information. Popular search engines like Google, Bing, and Yahoo have revolutionized how we access information, leveraging sophisticated algorithms and technologies to provide accurate and relevant search results [21].

History

The history of search engines dates back to the early 1990s. The first search engine, Archie, was created in 1990 by Alan Emtage, a student at McGill University. Archie indexed FTP archives, allowing users to find specific files . In 1993, the World Wide Web Wanderer, the first web robot, was developed by Matthew Gray to measure the growth of the web .

By 1994, several search engines had emerged, including WebCrawler, which was the first to index entire web pages. In 1996, Larry Page and Sergey Brin developed Backrub, which later became Google. Google's PageRank algorithm, which ranked web pages based on their link structure, significantly improved search result relevancy and set a new standard for search engines .

Over the years, search engines have continued to evolve, incorporating advanced technologies such as machine learning, artificial intelligence, and NLP to enhance their capabilities and provide more accurate and relevant search results [21].

Technological Advancements

Modern search engines use a combination of crawling, indexing, and ranking to deliver search results. Crawlers, also known as spiders, traverse the web to discover and index new content. The indexed content is then organized and stored in databases. When a user enters a query, the search engine retrieves relevant documents from its index and ranks them based on various factors, such as keyword relevance, content quality, and user engagement metrics .

Advanced search engines leverage NLP to understand the context and intent behind user queries, enabling more accurate and relevant search results. Machine learning algorithms are used to continuously improve search result accuracy by analyzing user behavior and feedback. Additionally, search engines use sophisticated ranking algorithms to prioritize the most relevant and high-quality content .

1.5 Information Retrieval

Information retrieval (IR) streamlines the efficient and effective extraction of relevant information from large sets of unorganized or partially organized data. IR systems help users locate, access, and present information that matches their search queries or information needs [21].

History

Information retrieval has its roots in libraries and archives, which started to organize and index their scholarly holdings as far back as the ancient city of Alexandria. By the 1800s, information processing had evolved to include concepts such as punch cards. In 1931, Emanuel Goldberg patented the ”Statistical Machine,” the first electromechanical document retrieval device [21].

As a scientific discipline, information retrieval began to develop during the mid-20th century. Gerard Salton and Hans Peter Luhn were pioneers in creating some of the earliest automated document retrieval models. In the 1960s, Salton's work at Cornell University on the SMART Information Retrieval System introduced key conceptual breakthroughs such as the term-document matrix, vector space model, relevance feedback, and Rocchio classification .

The 1970s saw the advent of advanced retrieval techniques and the development of probabilistic models. By the late 1990s, information retrieval systems and models became accessible to the general public outside of academic and institutional settings [21].

Types of Information Retrieval Models

Information retrieval models have evolved to address various challenges and enhance the retrieval of relevant information. Classical models set the foundation for the field, while non-classical models aim to overcome their limitations. Modern IR models incorporate state-of-the-art technologies such as machine learning and language models. The major categories of information retrieval models include :

- **Classical Information Retrieval Models**
  - **Boolean Model** The Boolean model is one of the oldest and simplest models used in information retrieval. It combines query terms through Boolean logic operators and treats documents as sets of terms. The model identifies documents that satisfy specific query conditions but does not rank or handle partial matches effectively .

In the Boolean model, queries are posed as Boolean expressions of terms, combined using AND, OR, and NOT operators. The Boolean AND of two statements x and y means both must be satisfied, resulting in a smaller or equal set of documents. The Boolean OR requires any one of the statements to be satisfied, leading to a larger or equal set of documents .

Google's search engine uses a two-stage approach for ranking web page results: first, a simple Boolean retrieval model produces an unordered set of matching documents, then a relevance estimator orders them [21].

- **Vector Space Model** The vector space model represents documents and queries as vectors in a multidimensional space, with each dimension corresponding to a term. Cosine similarities between the query vector and document vectors are calculated to retrieve relevant documents. This model addresses the Boolean model's limitation of not ranking results and is widely used for text retrieval .

  In the vector space model, term frequency-inverse document frequency (tf-idf) is a common method of weighting, highlighting the significance of frequently appearing words in a document that are scarce in the corpus .

  The model calculates document similarities by comparing the angles between query and document vectors, using cosine distance as a similarity metric to rank search results based on similarity to the query [21].
- **Probabilistic Model** The probabilistic model computes the probability of a document's relevance to a query based on parameters such as term frequency and document length. This model excels at issuing weighted statistics and ranking result outputs, making it suitable for large datasets .

  Probabilistic models account for uncertainties in understanding user information needs and estimating document relevance. Examples include classic probabilistic models like BIM, BM11, BM25, language models for information retrieval, and Bayesian networks-based IR models .

  The probabilistic relevance feedback mechanism refines initial search results by leveraging user feedback on the relevance of retrieved documents, leading to improved accuracy and relevance in subsequent queries [21].

- **Non-Classical Information Retrieval Models**
  - **Cluster Model** The cluster model groups similar documents into clusters, allowing for the retrieval of relevant documents based on the cluster to which a query belongs. This model enhances retrieval efficiency by reducing the search space .

    In the cluster model, documents are represented as vectors in a multidimensional space, and similarity metrics, such as cosine similarity, are used to measure the proximity between documents. Various clustering algorithms, such as k-means and hierarchical clustering, are employed to group similar documents together .
  - **Latent Semantic Indexing (LSI)** Latent Semantic Indexing (LSI) captures the latent relationships between terms and documents by reducing the dimensionality of the term-document matrix using techniques like Singular Value Decomposition (SVD). This model addresses synonymy and polysemy issues, improving retrieval accuracy .

    LSI represents documents and queries in a lower-dimensional space, where similar documents are closer together. This approach mitigates the effects of noisy and irrelevant terms, enhancing the retrieval of semantically related documents[21] .

- **Neural Network Model** The neural network model leverages artificial neural networks to model complex relationships between queries and documents. This model has gained popularity with the advent of deep learning techniques, enabling more accurate and context-aware information retrieval .

  In the neural network model, word embeddings represent words as dense vectors in a continuous vector space. Neural networks, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), process these embeddings to capture contextual information and semantic relationships .

  The Bidirectional Encoder Representations from Transformers (BERT) model, developed by Google, exemplifies the neural network model's potential in information retrieval. BERT's bidirectional training approach allows it to understand the context of words in a query, leading to more accurate retrieval results [21].

- **Reinforcement Learning Model** The reinforcement learning model employs reinforcement learning techniques to optimize the retrieval process by learning from user interactions. This model adapts to user preferences and behavior, continuously improving the relevance and quality of retrieved documents .

  In the reinforcement learning model, an agent interacts with the environment (search system) and receives feedback in the form of rewards or penalties based on the relevance of retrieved documents. The agent updates its policy to maximize cumulative rewards, leading to improved retrieval performance over time .

  Reinforcement learning algorithms, such as Q-learning and deep Q-networks (DQNs), are used to train the agent to make optimal decisions in the retrieval process. This approach enables personalized and context-aware information retrieval, catering to individual user needs and preferences [21].

- **Language Model** Language models predict the probability distribution of words in a text sequence, enabling more accurate information retrieval by understanding the context and semantics of queries and documents. These models have gained prominence with the rise of deep learning techniques .

  The language model calculates the probability of a word sequence by decomposing it into conditional probabilities of individual words given the preceding words. This approach captures the contextual dependencies between words, improving the relevance and coherence of retrieved documents .

  Pre-trained language models, such as BERT and GPT-3, have demonstrated significant improvements in various NLP tasks, including information retrieval. These models leverage large-scale pre-training on diverse text corpora, followed by fine-tuning on specific tasks to achieve state-of-the-art performance .

  The language model-based retrieval approach integrates query and document representations into a unified framework, allowing for more accurate matching and ranking of search results. This approach has been successfully applied to various IR tasks, such as question-answering, document ranking, and text summarization [21].

1.6 Applications in E-Learning

Information retrieval plays a critical role in e-learning by enabling efficient access to relevant educational resources. Advanced search systems leverage NLP and machine learning techniques to

enhance the retrieval process, providing personalized and context-aware search results .

In e-learning platforms, information retrieval facilitates the discovery of educational content, such as online courses, articles, videos, and interactive materials. By understanding user queries and preferences, search systems can recommend relevant resources, improving the learning experience .

Moreover, information retrieval techniques support various e-learning functionalities, such as automated grading, plagiarism detection, and content summarization. These applications streamline the educational process, allowing educators to focus on delivering high-quality instruction and learners to engage with relevant and engaging content .

1.7 Future Directions

The future of information retrieval in e-learning is poised to benefit from advancements in artificial intelligence, machine learning, and NLP. These technologies will enable more accurate and personalized search results, enhancing the efficiency and effectiveness of educational resource discovery .

Emerging trends, such as conversational agents and voice search, will further transform the e-learning landscape by providing intuitive and interactive search experiences. These technologies will enable users to engage with search systems naturally and seamlessly, improving the accessibility and usability of e-learning platforms .

Furthermore, the integration of information retrieval with other AI technologies, such as computer vision and reinforcement learning, will enable more sophisticated and context-aware search systems. These systems will be capable of understanding complex user queries, adapting to individual learning needs, and providing timely and relevant educational resources .

1.8 Natural Language Processing (NLP)

Natural Language Processing (NLP) focuses on the interaction between computers and human language, enabling machines to understand, interpret, and generate human language. NLP techniques are crucial in various applications, such as information retrieval, machine translation, sentiment analysis, and text summarization .

1.8.1 Overview

NLP involves the application of computational techniques to analyze and process natural language data. It encompasses various tasks, such as tokenization, part-of-speech tagging, named entity recognition, syntactic parsing, and semantic analysis .

Tokenization involves breaking down text into individual words or tokens, while part-of-speech tagging assigns grammatical categories to each token. Named entity recognition identifies and classifies named entities, such as names, dates, and locations, in the text .

Syntactic parsing analyzes the grammatical structure of a sentence, identifying relationships between words and phrases. Semantic analysis involves understanding the meaning and context of the text, enabling machines to interpret and generate human language accurately .

1.8.2 History

The history of NLP dates back to the 1950s when researchers began exploring machine translation and language understanding. Early approaches relied on rule-based systems, which involved manually crafted rules for language processing .

In the 1980s and 1990s, statistical methods gained prominence, leveraging large corpora of text data to train machine learning models for NLP tasks. The introduction of the Hidden Markov Model (HMM) and the development of the first statistical machine translation system marked significant milestones in NLP history .

In recent years, deep learning techniques have revolutionized NLP, enabling the development of sophisticated models that can process and understand natural language with unprecedented accuracy. Models like BERT, GPT-3, and Transformer-based architectures have set new benchmarks in various NLP tasks .

1.8.3 Applications

NLP has numerous applications across different domains, enhancing the capabilities of various systems and improving user experiences. Some key applications of NLP include:

- **Information Retrieval** NLP techniques enable more accurate and relevant information retrieval by understanding the context and intent behind user queries. Search engines leverage NLP to process and interpret queries, improving the quality and relevance of search results .
- **Machine Translation** Machine translation systems use NLP to translate text from one language to another. These systems have evolved from rule-based approaches to statistical and neural machine translation models, significantly improving translation accuracy and fluency .
- **Sentiment Analysis** Sentiment analysis involves determining the sentiment or emotion expressed in a piece of text. NLP techniques analyze text to identify positive, negative, or neutral sentiments, enabling businesses to understand customer opinions and feedback
- **Text Summarization** Text summarization involves generating concise summaries of long documents while preserving key information and context. NLP techniques, such as extractive and abstractive summarization, enable the automatic generation of summaries for various types of text data .
- **Named Entity Recognition (NER)** NER identifies and classifies named entities, such as names, dates, and locations, in the text. This technique is used in various applications, such as information extraction, question-answering systems, and content categorization .

1.8.4 Future Directions

The future of NLP is poised to benefit from advancements in deep learning, transfer learning, and unsupervised learning techniques. These technologies will enable more accurate and context-aware language processing, improving the performance of NLP systems across various tasks .

Emerging trends, such as conversational AI and voice-based interfaces, will further transform the NLP landscape by providing more intuitive and interactive user experiences. These technologies will enable seamless interaction between users and machines, improving the accessibility and usability of NLP systems .

Moreover, the integration of NLP with other AI technologies, such as computer vision and reinforcement learning, will enable more sophisticated and context-aware language processing. These systems will be capable of understanding complex language nuances, adapting to individual user needs, and providing timely and relevant information .

1.9 State of the Art in NLP Techniques

1.9.1 Keyword Extraction

Keyword extraction is a fundamental task in NLP that involves identifying the most relevant and informative words or phrases in a text. These keywords provide a concise summary of the content, enabling efficient information retrieval and content categorization .

Various techniques are used for keyword extraction, ranging from rule-based approaches to machine learning and deep learning methods. Some popular keyword extraction techniques include:

- **Term Frequency-Inverse Document Frequency (TF-IDF)** TF-IDF is a statistical measure that evaluates the importance of a word in a document relative to a collection of documents (corpus). The term frequency (TF) represents the frequency of a word in a document, while the inverse document frequency (IDF) measures the rarity of the word across the corpus .

  The TF-IDF score is calculated as the product of TF and IDF, highlighting words that are frequent in a document but rare in the corpus. This approach helps in identifying the most relevant keywords that capture the essence of the document .

- **Rapid Automatic Keyword Extraction (RAKE)** RAKE is an unsupervised algorithm that identifies keyword phrases in a text by analyzing the co-occurrence of words. The algorithm partitions the text into candidate keywords based on word delimiters, such as spaces and punctuation marks .

  Each candidate keyword is assigned a score based on the frequency and co-occurrence of words within the phrase. The highest-scoring candidate keywords are selected as the most relevant keywords for the text .

- **TextRank** TextRank is a graph-based ranking algorithm inspired by the PageRank algorithm used in web search engines. In TextRank, words or phrases are represented as nodes in a graph, and edges between nodes represent the co-occurrence relationships .

  The algorithm iteratively updates the importance score of each node based on the scores of its neighboring nodes. The highest-scoring nodes are selected as the most relevant keywords for the text .

- **KeyBERT** KeyBERT is a keyword extraction technique that leverages BERT embeddings to identify the most relevant keywords in a text. BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained language model that captures contextual information and semantic relationships between words .

  KeyBERT uses BERT embeddings to generate dense vector representations of words and phrases in the text. The algorithm calculates the cosine similarity between the document and candidate keywords, selecting the highest-scoring keywords as the most relevant .

1.10 Applications in E-Learning

Keyword extraction plays a crucial role in e-learning by enabling efficient content discovery, summarization, and categorization. E-learning platforms leverage keyword extraction techniques to enhance the searchability and accessibility of educational resources .

In e-learning, keyword extraction is used to:

- **Improve Searchability** Keyword extraction enhances the searchability of educational resources by identifying relevant keywords that represent the content. This enables users to discover relevant resources based on keyword searches, improving the efficiency and effectiveness of the search process .
- **Generate Summaries** Keyword extraction techniques generate concise summaries of educational content by identifying the most informative words or phrases. These summaries provide learners with a quick overview of the content, helping them decide whether to engage with the material .
- **Categorize Content** Keyword extraction facilitates the categorization of educational resources by identifying keywords that represent the content. This enables the organization of resources into relevant categories, improving the accessibility and navigation of e-learning platforms .

1.10.1 Future Directions

The future of keyword extraction in e-learning will benefit from advancements in deep learning, transfer learning, and unsupervised learning techniques. These technologies will enable more accurate and context-aware keyword extraction, improving the relevance and quality of extracted keywords .

Emerging trends, such as zero-shot learning and few-shot learning, will further enhance keyword extraction by enabling models to generalize to new and unseen data with minimal supervision. These techniques will enable efficient keyword extraction in diverse and dynamic e-learning environments .

Furthermore, the integration of keyword extraction with other NLP tasks, such as topic modeling and sentiment analysis, will enable more sophisticated and context-aware content analysis. This will improve the searchability, summarization, and categorization of educational resources, enhancing the overall e-learning experience .

Chapter 2

Conceptual Study

2.1 Introduction

In the preceding chapter, we delved into the dynamic landscape of Massive Open Online Courses (MOOCs), exploring their various approaches, related concepts, and emerging trends in the educational domain. We discussed the pivotal role of MOOC platforms in democratizing access to education and facilitating diverse learning experiences for individuals worldwide.

This chapter shifts our focus towards the design aspects of MOOC platforms, acknowledging their multifaceted nature comprising numerous components aimed at optimizing the learning journey. MOOCs encompass a spectrum of functionalities, including but not limited to search engines, recommendation systems, interactive forums, and assessment tools, each playing a vital role in enhancing user engagement and knowledge acquisition.

Within the scope of this thesis, our primary emphasis lies on the development and implementation of an AI-based search engine tailored specifically for MOOC environments. While MOOCs integrate various components to offer comprehensive learning experiences, the search engine stands out as a critical element facilitating efficient course discovery and navigation for users. As such, our research endeavors to explore innovative approaches and methodologies to enhance the search capabilities within the MOOC ecosystem, leveraging the power of artificial intelligence and machine learning techniques.

By focusing on the search engine component, we aim to address key challenges and opportunities in course exploration and recommendation within MOOC platforms. Through a systematic exploration of advanced AI-driven methodologies, including Long Short-Term Memory (LSTM) networks, Graph Neural Networks (GNNs), and other cutting-edge techniques, we seek to optimize search relevance, accuracy, and user satisfaction. Subsequent sections will elucidate our approach towards designing and implementing an AI-enhanced search engine within the context of MOOCs, providing insights into the operational framework and implementation strategies employed in our research.

2.2 System design

A well-built advanced search system must be developed in a methodical way, using different methodologies while ensuring that the search results remain relevant and accurate. By guiding them through distinctive steps that are shaped according to diverse aspects of the search process and the user's query it enhances the searching experience for users:

1. **User Query Input:** Users input their queries into the system, specifying their interests along with customizable filters to tailor their search.
2. **Query Preprocessing:** The system processes the query in two steps. It first corrects the term for some syntactic errors. If a term is misspelled, the system suggests corrected versions based on similarity of the term with important terms in the dataset. Coherence measures are used to select the most appropriate correction. The query is then lemmatized and stop words are removed. It retains logical operators like ”and” and ”or” for further processing. Normalized and unnormalized versions of the query are saved to satisfy requirements of different components in the system.
3. **Understanding User's Query:** This step is highly crucial because the structure and intent of the query is understood. It first attempts to recognize logical structure using the rule-based approach with operators like ”and” and ”or”. Topic modelling recognizes the topic probability distribution of the query; then, clusters of courses with similar distribution are selected. Query keyword extraction further narrows down the query. After that, keywords from selected clusters are semantically matched. Named entity recognition/disambiguation identifies important entities in the query for further query refinement.
4. **Query Enhancement:** The system always assumes that users do not express exactly what they need and they are not accurate; therefore, query enhancement mechanisms are used. Unimportant keywords are replaced by important ones, and related keywords are appended to the query to enhance search results.
5. **Machine Learning Relevance Evaluation:** The textual data of the filtered courses and the refined query is passed to a machine learning model or a ranking model for query evaluation in terms of final relevance of the document to enhance the precision of search.
6. **Course Filtering:** Last but not least, courses are filtered on the basis of popularity, availability, and rating to present users with the most relevant and desirable options.
7. **Feedback Loop:** The system includes a feedback mechanism that allows users to provide feedback on the search results. Users can indicate that the results do not match their queries or if any results do not have relevance (negative feedback) or grade and approve the search results (positive feedback). This feedback is used to constantly improve the accuracy and relevance of the system.

By integrating a variety of methodologies and precise processing steps, the advanced search system adeptly handles the complexities of user queries, providing refined and relevant results. This strategic integration of techniques enhances search precision, ensuring an intuitive and effective search experience for users.

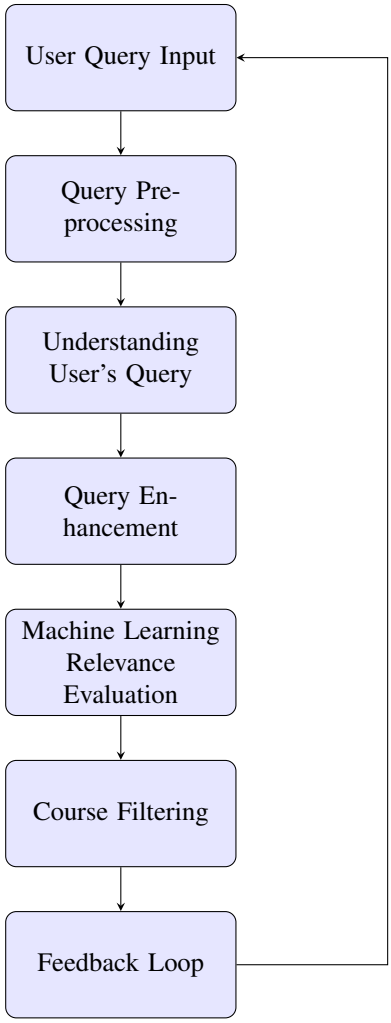


Figure 2.1: Overall system design

2.3 Data source:

the dataset used in developing this system was obtained from *Kaggle*, *Kaggle* is a data science competition platform and online community of data scientists and machine learning practitioners under Google LLC. Kaggle enables users to find and publish datasets, explore and build models in a web-based data science environment, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges.

2.4 Dataset Characteristics:

the dataset contains 12000 samples or rows, with each row containing textual data representing different attributes of the course, these attributes include:

- Title: The title of the course.
- Description: A detailed description of the course content.

- Cost: The cost or price of the course.
- Availability: Information about the availability of the course.

2.5 Data preparation:

Textual data in natural language processing (NLP) often contains a lot of noise, which can disrupt tasks or affect the accuracy of results. Therefore, preprocessing steps are crucial to ensure that only relevant data is kept for analysis. The following preprocessing steps were applied to the dataset to address this challenge:

2.5.1 Handling null values:

one of the most important and initial preprocessing steps is identifying and handling rows with null values, and since our main focus is on the ”Title” and ”Description” columns and due to the inherent complexity of interpolating textual data, rows with null values in these columns were simply removed.

2.5.2 Tokenization:

Another important preprocessing step to prepare the data for further tasks is tokenization. Tokenization, There are different types of tokenization; for our case, we will use word tokenization, using spaces as clear boundaries between the tokens.

2.5.3 Removing noise:

another crtitical preprocessing step is removing words or terms that do not carry any meaning, which often refered as noise, this step ensures that only relevant information is kept for further analysis and processing, Noise often consists of words that are not grammatically or syntactically correct and do not carry any meaning, typically due to human errors or issues with data storage.

the challenge however is in determining if a word is considered noise, a straightforward approach would be to check if a word is a valid english word but this approach doesn't perform good with domain-specific terms, an example of that is C++, therefore we are going to follow this approach:

1. If a word has a frequency equal to or greater than the specified threshold, it is considered a valid word.
2. If not the word is checked to see if it's a valid english word.
3. If not a search using wikidata is performed on the word, If there's results the word is kept.
4. If none of the above conditions are met the word is considered noise and removed.

2.5.4 Handling Irrelevant Words

This step focuses on identifying words that are considered valid English words but do not carry or add any meaning to our analysis. These are commonly called stop words. In addition to the standard English stop words, the task here is to identify domain-specific words that also do not add any significant meaning to our task.

To address this, we use a technique called Term Frequency-Inverse Document Frequency (TF-IDF), a numerical statistic that reflects the importance of a word in a document. It is commonly used in NLP to represent the relevance of a term to a document or a corpus of documents. The TF-IDF algorithm takes into account two main factors: the frequency of a word in a document (TF) and the frequency of the word across all documents in the corpus (IDF)[37].

We experiment with different thresholds, finding words that have low TF-IDF scores below the threshold. We find the appropriate threshold by observing the results and saving the results for further processing.

2.6 Topic modeling

Working with textual data in machine learning is complex due to the inherent unstructured nature of the data. To understand the underlying structure of the data, we use topic modeling.

2.6.1 Overview of topic modeling

topic modeling is an unsupervised statistical method for discovering abstract 'topics' that exist within a collection of documents. It scans or 'mines' text to detect frequently used words or phrases and groups them to provide a summary that best represents the information in the document[6].

2.6.2 Model selection

For our topic modeling, we chose `BERTopic`. BERTopic is a topic modeling technique that leverages transformer-based models and c-TF-IDF to create dense clusters allowing for easily interpretable topics while keeping important words in the topic descriptions[26].

We chose BERTopic over the other traditional methods like Latent Dirichlet Allocation (LDA) because of its relies on transformer-based models, which showed significant results in understanding the complexities of natural language. BERTopic uses these models to generate embeddings that capture the semantic meaning of the text, and then uses these embeddings alongside clustering algorithms that allows for accurate and efficient retrieval of topics. LDA on the other hand despite being the most used approach in topic modeling, it relies on straightforward probabilistic modeling that can fail capture the complexities of language.

Another reason for considering BERTopic is its out of the box support of online topic modeling or what's called incremental topic modeling, is the ability to iteratively learn and update a topic model using new, small batches of new data. Essentially, such an approach allows the topic model to be refreshed and improved with the incoming data, even in a case where this new information was not seen beforehand to train the model. This makes online topic modeling particularly useful for dynamic and evolving datasets where new topics and trends may emerge over time, this is useful in our case when new courses gets added to the system causing the emerge of new topics[25].

2.6.3 How BERTopic Works:

BERTopic can be viewed as a sequence of steps to achieve its topic representation.

- Embedding: the documents or the textual data are converted to embeddings, An embedding is a low-dimensional representation of high-dimensional data, which makes the data more manageable for machine learning tasks. embeddings capture the semantic meaning of the data by putting the represenation of similar and related data relatively closer in the embedding space, in our case the text data that represent the title and the description of the courses are converted into embeddings using a sentence transformer which is a pretrained model optimized for semantic similarity, BERTopic by default uses "`all-MiniLM-L6-v2` which is English language model trained specifically for semantic similarity tasks which works quite well for most use cases[8]. "

- Dimensionality reduction: Clustering methods struggle with high dimensional data this is known as "the curse of dimensionality" which refers to the various challenges and complications that arise when analyzing and organizing data in high-dimensional spaces,as the number of the dimensions in the data increases this causes the data to become sparse this will result in increased computations and any processing relies on extracting meaningful information from the distances between the data points will become less useful as the distance between data points in high dimensional spaces tends to decrease[17], to address this issue a dimensionality reduction method is used, there are different approaches to dimensionality reduction such as `PCA`, by default BERTopic uses UMAP which is a new technique by McInnes et al, and the reason for that UMAP tends to better preserve the global structure of the data. This can be attributed to UMAP's strong theoretical foundations, which allow the algorithm to better strike a balance between emphasizing local versus global structure[14] It is important to maintain this structure because it contains crucial information required to form clusters of documents that contain related semantic features.

- Clustering: after that a clustering algorithm is used, specifically HDBSCAN which is a a clustering algorithm used in unsupervised learning to identify groups of similar data points known as clusters, HDBSCAN is an extension of the popular DBSCAN algorithm to address its limitations[18] and has a nice feature of identifying outliers which are data points that do not fit into any cluster.

- Bag of words: In the text representation step of BERTopic, we have a method for handling the various shapes and sizes of clusters. Instead of relying on fixed assumptions about clusters, we combine all documents in each cluster into one large document. Then, for each cluster, we count the occurrences of each word. This gives us a bag-of-words representation for each cluster. This representation focuses on the importance of words within topics. We adjust for variances to make sure that this word representation works well regardless of the cluster's size. This flexibility in approach highlights the capability of BERTopic to capture diverse topics in a non-constraining manner regarding cluster structures.

- Topic representation: For effective representation of topics, the conventional TF-IDF approach is then modified to take into account clusters rather than individual documents. This modification is known as class-based TF-IDF, where documents are regarded as one entity within a cluster. Therefore, applying the conventional TF-IDF to these clusters computes importance scores for the words for each topic. Here, the words bearing more importance to each cluster are surfaced, thus describing topics effectively. To represent the class-based TF-IDF, the idea is to convert each of the clusters as one document; then, the frequency of each word within that cluster is extracted. This representation is then L1-normalized so that variations in topic

sizes are taken into account. Besides, the conventional TF-IDF is further developed by including a logarithmic transformation for fine-tuning the importance of scores of the words in the clusters. The modified approach avails effective representation of the topics compared to the standard TF-IDF algorithm.

2.6.4 Preprocessing:

In addition to the previous preprocessing steps mentioned in the previous section, no further preprocessing is performed specifically for BERTopic. It is advised not to remove stop words, as transformers rely on them to provide accurate context embeddings. Removing stop words before generating the embeddings can affect the performance of the model.

We combine the ”Title” and ”Description” columns into a single string for each row.

2.6.5 Implementation:

We use topic coherence as a guide to choose the parameters for BERTopic.

2.7 Spelling correction

2.7.1 Overview

Spelling correction is a language processing tool designed to detect and correct common spelling errors made by users when entering their queries

2.7.2 Approaches

We suggest two possible approaches:

- **Neural network:** In this case the network would take the query that contains the misspelled terms and the network will output the corrected version of the query.
- **Word embedding:** We use a word embedding model, A word embedding is a way to represent words as vectors in a multidimensional space where the distance between them in this space can be interpreted as semantic similarity and the relationship between words[29]. In this case word embedding is used as a way to calculate a coherence score that can be used to choose the right replacement for the misspelled word or term.

Due to the complexity of the first approach and the the limited time we use the second approach which include using a word embedding model as follows:

1. **Identifying misspelled words:** We identify misspelled words by checking if they are considered valid english words or domain-specific terms from an inverse-index of the important and domain-specific terms from the dataset, if none of the previous conditions are met the word is considered misspelled.
2. **Generating candidates:** to generate candidates we use Levenshtein distance which is the minimum number of edit operations required to convert one string to another, these edit operations include insertions, deletions, and substitutions[24]. Words with low edit distance are considered good candidates for the misspelled word.

3. **Calculating coherence score:** for each candidate we calculate the embeddings for each word or term in the query, then we take the average of the embeddings to get a coherence score for each candidate.
4. **Candidate selection:** In this step we simply choose the candidate with the highest score.

Now we could train our own word embedding model using our dataset, but there's one big problem which what if the user enters terms or words that doesn't exist in the dataset. to address this issue instead we use a pretrained word embedding model, then we fine tune it on our dataset.

2.8 Text clustering:

2.8.1 Overview:

Clustering can also be identified as cluster analysis-an unsupervised machine learning technique applied in grouping homogeneous data points into clusters. In its literal meaning, a cluster is thus a collection of data points that have similar characteristics or, inversely, it is an assemblage of data points, wherein a data point takes on the same qualities as the other data instances in the same cluster and diverges from the rest of the data points in a different group. This is done in an unsupervised way, which means it does not depend on labeled data, but the process uses characteristics of the data itself to guide the process.[4].

2.8.2 Types of clustering methods:

There are various methods to clustering. each different approach is tailored to a specific data distribution and use cases:

- **Centroid-based clustering:** also known as partition clustering is a straighforward clustering method. It groups data by comparing their proximity to chosen central values and breaks down the dataset into a group of clusters, each represented by a vector. A prominent algorithm in this category is the K-Means. However, although very simple, it becomes difficult to determine the ideal number of clusters. This approach is commonly applied in cases involving a relatively large volume of data, where it calculates the distance between all clusters and centroids at each iteration using metrics such as the Euclidean or Manhattan distances.
- **Density-based clustering:** follow a different route from distance-based approaches in that they prioritize data density rather than distance. In this method, clustering is done based on the regions with high data concentration, therefore allowing for clusters of arbitrary shapes and sizes with maximal homogeneity. It effectively includes noise and outliers in the dataset, thus making the method robust in the face of real-world data, an example of this is the DBSCAN algorithm.
- **Didstribution-based clustering:** uses probability distributions to group data points based on the likelihood that they come from the same distribution. Each cluster is defined by a central point, with data points further from this point having lower probabilities of inclusion. Compared to both density- and boundary-based methods, this distribution-based approach does not directly require cluster shape specification a priori; however, hyperparameters are used for tuning, which, if inconsistent, may lead to undesirable outcomes.

Each clustering method has its own advantages and use cases[34].

2.8.3 Feature extraction:

The clustering is not based on the actual textual data of the courses but instead of the topic probability distribution of the documents resulted from BERTopic, each document is represented by a vector where the size of each vector represents is the number of topics providing a compact representation of each document.

2.8.4 Selection of the clustering algorithm:

To decide which clustering algorithm to use, we first visualize the distribution of the data, in our case it's the vector that contains the topic probability distribution of the textual data of the courses.

Knowing that our data is basically vectors that contains the topic probability distribution, traditional distance metrics like Euclidean distance or cosine similarity may not capture the underlying distribution and true similarity between the data points, for this a different measure is used. Jensen-Shannon Divergence (JS Divergence) is a symmetric measure of divergence between two probability distributions. It was introduced by Barry E. S. Lindgren in his 1991 paper "Some Properties of Jensen-Shannon Divergence and Mutual Information"; it generalizes the Kullback-Leibler (KL) divergence to be asymmetric.[33].

Jensen-Shannon (JS) divergence is calculated by following these steps:

Average Distribution: First we calculate the average distribution M of the two distributions P and Q:

$$M = \frac{1}{2}(P+Q) \tag{2.1}$$

KL Divergence Calculation: Then we calculate the KL divergence P to M and from Q to M:

$$D_{KL}(P \parallel M) = \sum_i P(i) \log \frac{P(i)}{M(i)} \tag{2.2}$$

$$D_{KL}(Q \parallel M) = \sum_i Q(i) \log \frac{Q(i)}{M(i)} \tag{2.3}$$

Weighted Average: Then we take the average of these two KL divergences:

$$D_{JS}(P \parallel Q) = \frac{1}{2} D_{KL}(P \parallel M) + \frac{1}{2} D_{KL}(Q \parallel M) \tag{2.4}$$

After that, we calculate a similarity matrix between the topic probability distributions of every textual representation of the courses using Jensen-Shannon divergence and every other course. Since Jensen-Shannon calculates the divergence between two probability distributions, we take $1 - D_{\text{JS}}$ where D_{JS} is the JS score. The next step is to visualize the data points based on the similarity matrix. This is achieved using a dimensionality reduction algorithm called t-SNE (t-distributed Stochastic Neighbor Embedding). On its input is a similarity matrix. The outcome of T-SNE is a two-dimensional embedding, where the distance of these embeddings captures the similarity between the topic distributions of each textual representation of each course.[13].

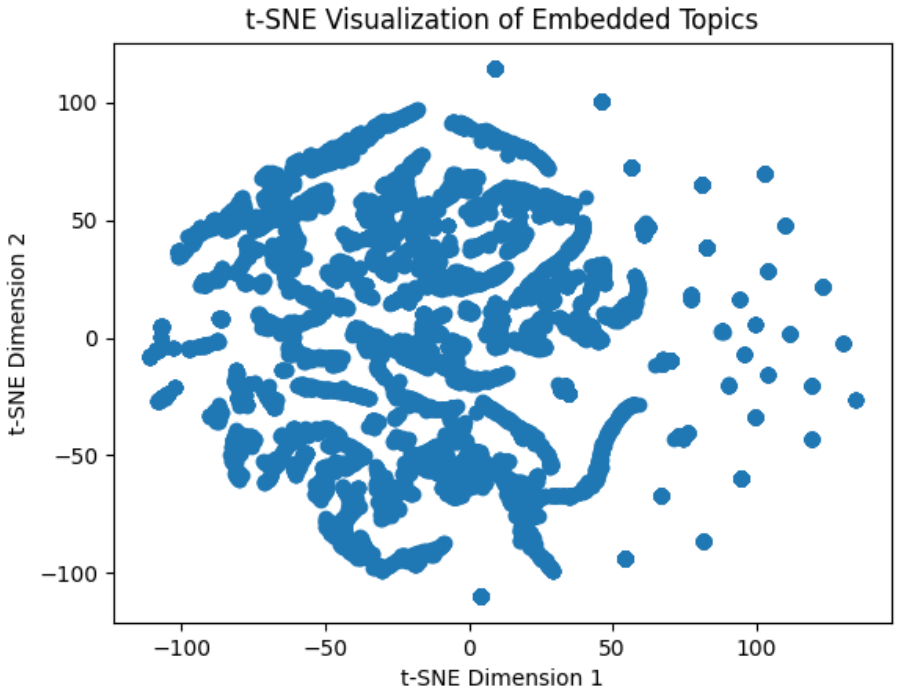


Figure 2.2: Text clustering using T-SNE based on topic distribution

After visualizing the data it's clearly obvious the data is following various shapes and separation. Some of them follows linear formations and spherical groupings of data points. Based on that a density-based clustering algorithms seems the most suitable choice for this task, where the concept of mean may not effectively capture irregular data distributions, density-based algorithms like DBSCAN that we are going to go with for this task rely on the concept of density rather than distance, **Density-Based Spatial Clustering of Applications with Noise (DBSCAN)** is a base algorithm for density-based clustering that can discover clusters of different shapes and sizes in large amounts of data, including noise and outliers[7].

The following are the two parameters used by the DBSCAN algorithm:

- **minPts:** The threshold value for the minimum number of points clustered together for a region to be considered dense[7].
- **eps (ε):** A distance measure which would be used to find the points in the neighbourhood of a point[7].

2.9 Named entity recognition\disambiguation

2.9.1 Introduction

Named Entity Recognition (NER) is the process used to identify specific categories of elements from within a text, be it names of individuals, organizations, locations, or time expressions, as well as quantities, among other types of entities that may have been defined. NER is among the basic tasks in natural language processing (NLP). In other words, NER goes through a text string, be it

a sentence, paragraph, or a whole document, by identifying the entities and classifying them into one type or class that has been predetermined for that category.[5].

Named Entity Disambiguation (NED) is one of the critical areas of research in Natural Language Processing (NLP), focusing on connecting a mention in a text entity with its corresponding entity in the knowledge base, such as a node in a knowledge graph. Accurate NED is essential for modern technology companies to connect documents, reports, press releases, and all other textual content to the relevant context stored in knowledge bases—boosting entity understanding for both academic research and business applications[39].

2.9.2 NER\NER techniques:

There are three approaches for named entity recognition and disambiguation:

1. **Rule-based approaches:** This method can be applied to the text to identify entities depending on their structural and grammatical properties. Examples include Named Entity Recognition using grammatical rules in the particular language of concern. These methods, however, have proven effective in practice and are, in fact, widely used. However, these methodologies can sometimes be very time-consuming and fail to generalize to unseen data[5].

2. **Machine learning approaches:** hese entail training AI-driven models with labeled datasets over both conditional random fields and maximum entropy, among other methods. Such traditional methods are known to include decision trees and support vector machines. In advanced methods, there entail recurrent neural networks and transformers. Such techniques are usually seen to have better generalization properties to unseen data, though they require a large labeled dataset and become computationally expensive[5].

3. **Hybrid approaches:** These combine both the rule-based and the machine learning to maximize on the two. They use the rule-based systems to get simple entities in order for it to quickly capture simple entities and thus utilize the machine learning systems to get complex entities.[5].

2.9.3 Data aquisition:

To create a dataaset to train an nermodel we use a pretrained ner model from spacy for the purpose of identifying possible named entity, and in addition to that we use keywords extracted from the previous section using keyBERTthis will representat out named entities from our dataset specifically the textual data that representat the title and the description of the courses.

After spotting the named entities, we do an entity search in Wikidata for entities that have the name as a label or as an alias of the entity mentioned in the text. The challenge now arises in determining which specific named entity from the candidates that really represents the named entity mentioned in the textual data.

To address this, we employ two measures of similarity:

- **Topic probability distribution similarity**: We use the topic probability distribution resulted from BERTopic for that specific course (its textual data); we extract textual data from each candidate, and we use the topic model to infer the topic probability distribution of the textual data of the entity. We calculate the similarity score using Jensen-Shannon by taking $1 - D_{\mathrm{JS}}$, where D_{JS} is the Jensen-Shannon score.

- **Semantic similarity**: In this method, we used a pretrained SBERT model for generating embeddings that capture the semantic meaning of the text. To represent the mentioned entity in the text, we used its surrounding words according to a specific window size. Then, we generated its embedding with the SBERT model. The same thing is done for the candidate entities by feeding their textual data to SBERT. Once the embeddings are derived, the similarity score is calculated by conducting the cosine similarity of the embedding of the mentioned entity and the embeddings of the candidate entities from Wikidata.

As for the categories that are used as labels to train the named entity recognition model, we use the information from the ”instance of” relation from the corresponding similar entity among the candidate entities. We use the label of the entity that the corresponding entity is an instance of as a label or category.

2.9.4 Identified issues:

Since the dataset is created based on the mentioned entities in the textual data of the courses, and knowing that these courses or materials discuss different topics and subjects in various domains, it's important to recognize that the development of knowledge in different domains is not uniform. Some domains would achieve a significant progress and advancements compared to others. As a result, on an e-learning platform where the goal is to provide knowledge about different domains, there would be imbalances in the dataset, where specific courses containing more named entities about a specific subject or domain compared to the others.

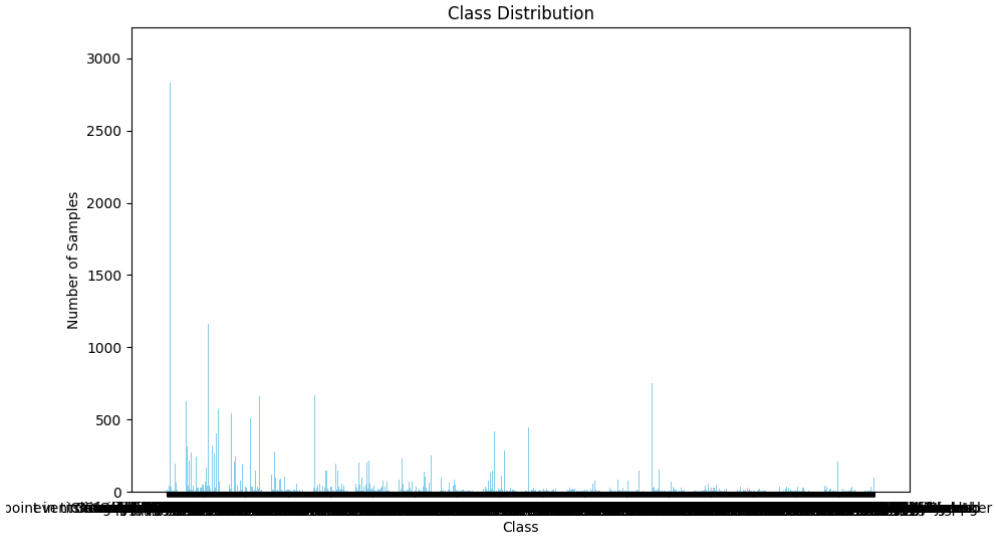


Figure 2.3: Named entity recognition dataset class distribution‘

These imbalances, in turn, would make the dataset that we are creating for NER biased, wherein instances in some classes outnumber the others. A model learnt on this type of imbalanced dataset would picks up the bias and then produce inaccurate, unreliable results even when the results seems to be promising.

to address this issue, we consider various existing solutions since imbalanced datasets are an endemic problem is the realm of machine learning:

- **The rebalancing methods:** include modifying the original dataset to get a more well-balanced class distribution either by oversampling the minority class or undersampling the majority class. Among the oversampling techniques are random oversampling, synthetic minority oversampling technique(SMOTE), and adaptive synthetic known as ADASYN. On the other hand, undersampling techniques include random undersampling, nearmiss, and tomek links[20].
- **Data augmentation**: Data augmentation It involves creating more data points artificial modification of available data. since we are dealing with natural language various transformation like random deleting and swapping the words to increase the size of the minority classes[20].
- **Cost-sensetive learning:** This involves crafting a loss function that assigns greater weights to the minority class, thereby prompting the model to allocate more attention to these less-represented classes.
- **Clustering**: We propose another method since we constructed the data from a knowledge graph where there's hierarchical relationships between the entities, we can try to follow a different method to address this imbalance by using clustering, specifically hierarchical clustering where we try to cluster the instances of minority classes with the majority classes, where we represent the named entities as embeddings where we use BERT to produce semantic embeddings and we use a hierarchical clustering algorithm to merge coherent minority classes with the majority classes. this should help in reducing the effect of the imbalance.

Based on the data distribution of the instances according to their classes, we can see that there is a significant imbalance where some classes reaches to approximately 5000 instances in the data points, and other classes contains only one instance. Given this significant difference between the different classes based on distribution of the instances, we propose a combination of undersampling and oversampling. we use undersampling to reduce the majority class size. We can use random undersample for that. whereas for oversampling, we use SMOTE, that is Synthetic Minority Oversampling Technique to increase the number of instances in the minority classes[20].

2.9.5 Data preparation:

The next step after acquiring the data is to prepare it since each named entity is represented or fed into the model along with its surrounding words according to a specified window size. It will accentuate the ability to disambiguate: by considering the context in which the named entity appears, even if we feed the model two sentences such as "apple is a fruit" and "apple is a phone company," the model can still pick up the right label for the entity, though they bear the same name. Usage of the surrounding words as part of the input helps the model in the extraction of representations that can capture the context in which the named entity appears. This is needed to help the model know the right entity on which to focus at data prediction time, since the same sentence or text can be fed a second time with a different named entity target. In order to achieve this, the named entities are marked with special tokens, like [START] and [END]. Next, we provide pairs of data. The first element of that is the piece of the text which contains the named entity marked with the previous special tokens. The second element is the label, or specifically the ID, of the entity of which the chosen entity from the Wikidata is an instance, and these pairs are then to be used to train the model.

2.9.6 Model architecture

For this task we chose to use Long Short-Term memory LSTM network due to its ability to capture sequential patterns and understand the context of the text alongside an attention mechansim that helps the model to focus on the target entity in the text, and because of some challenges with data collection process. Due to some challenges faced during acquiring and building the data required for the task, the number of the classes reached more than 1000 class, which can cause the training process infeasible if the data is too large this because trying to reduce the number of classes requires extensive traversal of the wikidata knowledge graph and due to the time and hardware constraints, two approaches to implement the model are proposed below. both approaches share the next first layers of the model:

1. **Embedding layer:** The input text is fed into the embedding layer that will convert the textual data into dense vectors.
2. **LSTM layer**: the embeddings are fed into the LSTM layer which is the core of the network that will help the model in understanding the sequential patterns and context of the input.
3. **Attention mechanism**: to help the model focus on the target entity an attention mechanism is added into the model it works by assigning varying degrees to different parts of the input helping the model to focus on the important parts of the input.
4. **Fully connected layer:** the output of the LSTM layer processed by the attention mechanism is process through a fully connected layer.

The difference in both approaches is basically the output layer and how the output is interpreted:

1. **Approach 1: Traditional classification**:
    - **Output layer(Softmax):** in this case the previous fully connected layer will have the same size as the number of the classes where it produces a vector that represents raw scores or logits for each class. after that a softmax activation function is applied to the output of the dense layers transforming the raw scores into probability distributions that represents the likelihood of the input data point belonging to a specific class.

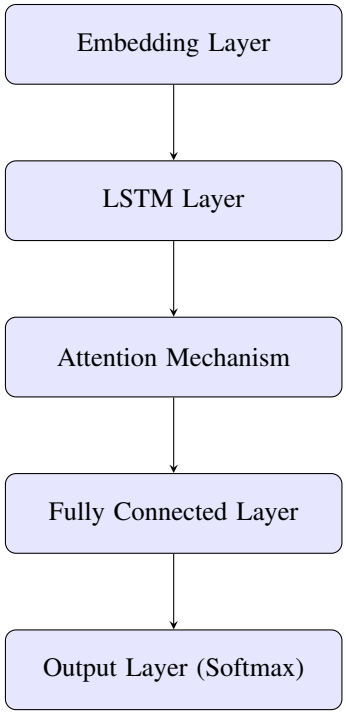


Figure 2.4: Named entity recognition: traditional classification model

2. **Approach 2: Representation learning**

   In this representation learning approach, it is not a typical classification problem anymore but more about learning representations. The network will try to produce an embedding by which classes are guided. The representations of the classes are either initialized as fixed representations of vectors in the embedding space or representations are directly learned. With the constant approach, each class has an associated fixed vector representation, and with the learnable approach, these representations are learned during training.

   - **Fixed representation** In the constant approach, every class is initialized with a fixed vector representation in the embedding space. This fixed representation remains invariant over the training process. The objective of the network would be to map the input embedding into closer proximity to its corresponding fixed class vector representation.

     A great challenge also lies in making sure a uniform class distribution in the embedding space. One is required to prevent overlap between classes and ensure that the distribution of class representations is even. Such a uniform distribution is important for learning and generalization and to avoid biases that might occur if there were an overdependence on a set of classes or an overlap in the class representations.

     Some of the common initialization methods for setting the initial values of the embedding vectors are:

     – **Random Initialization:** The embeddings for classes are initialized with random values sampled from a normal or uniform distribution.

     – **Orthogonal Initialization:** The class embeddings are initialized with orthogonal matrices, thus making sure that the embedding vectors are linearly independent.

This might prevent different dimensions of the embedding space from interfering with one another and make training more stable.

– **Random Initialization with Regularization:** A custom initialization method may be employed to guide the distribution of class embeddings while training the system. This procedure consists of using a standard initialization procedure, along with a regularization term, to drive the class embedding distribution to a configuration of interest. For instance, a regularization term can be added to the initialization procedure in order to drive a more uniform distribution of class representations in the embedding space and help avoid overlapping among the classes.

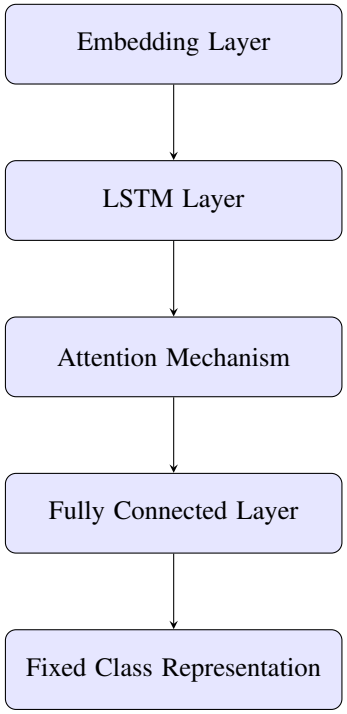


Figure 2.5: Named entity recognition: Representation Learning model architecture

- **Learnable representation:** For the learnable class representation, an extra network is added along with the static class representations to the existing architecture. The objective of this network is to serve in better learning by making both embeddings and class representations learnable parameters. It is pertinent to mention that this additional network is not used at inference time but is added to help the model learn better adaptive and effective representations during training.

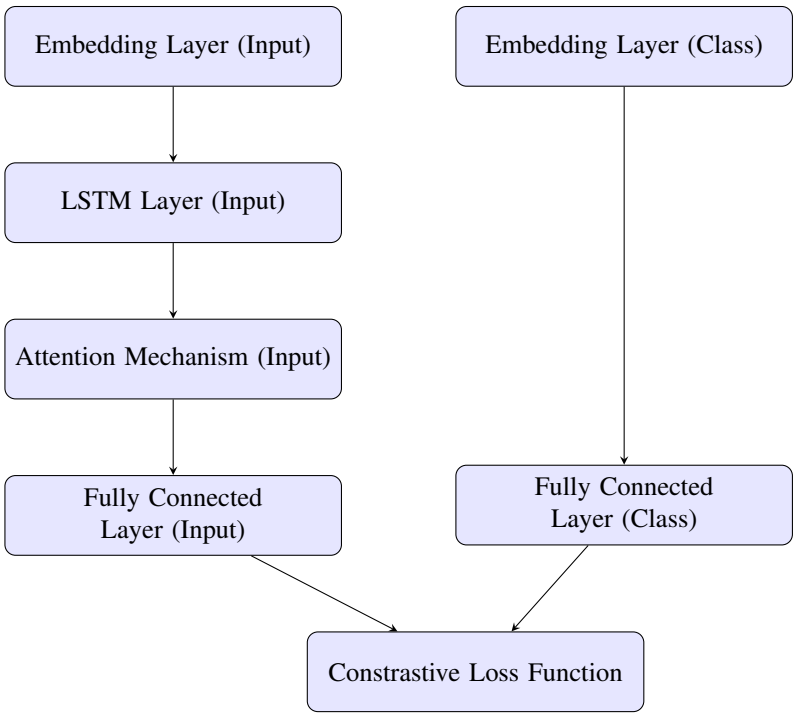


Figure 2.6: Named entity recognition: Representation learning(learnable approach)

The choice of the loss function depends on the chosen approach, for a classification problem a cross-entropy loss function is used which is suitable for multi-class classification problems and measure the difference between the predicted probabilities and the true labels in the data. for the second approach a different loss function is used here, since the goal is to map the embedding closer to its class vector representation this can be achieved using constrastive learning, a specific loss function used in constrastive learning is triplet loss which is a way to teach or guides the model into learning the similarities between data points or objects, and it relies on three components:

- **Anchor**: which is the output embedding representation of the input from the model.
- **Positive:** a data point or an embedding representation of a data point that is considered similar to the anchor in our case it's the class vector representation that corresponds to the input data.
- **Negative:** a data point or an embedding representation of a data point that is considered disimillar to the input or the anchor, in our case is class vector representation that doesn't correspond to the anchor.

enforces the distance between the anchor sample and the positive sample to be less than the distance between the anchor sample and the negative sample[16].

For the learnable approach, a different loss function is used. The scenario is one in which the goal is to map the embedding to its corresponding class vector representation via contrastive learning with a margin approach. This is contrastive learning, unlike triplet loss, which learns similarities of data using triplets of anchor, positive, and negative samples. In turn, contrastive loss directly maps corresponding class and embedding together.

This contrastive loss enforces a margin between embeddings of the same class and embeddings of different classes. This margin takes care that the embeddings of the same classes become nearer in the embedding space than embeddings of different classes

2.10 Keyword extraction

2.10.1 Introduction:

Keyword extraction, also known as keyword detection or keyword analysis, is a text analysis technique used for automatically determining and extracting the most prominent and relevant words and expressions within the content of a text. This is important for summarizing the content of texts, recognizing main topics, and so on.

Keyword extraction just makes it a whole lot easier to find relevant words and phrases, when it cones unstructured text data[1].

2.10.2 Keyword extraction techniques:

- Simple Statistical Approaches
  - Word Frequency: Lists the most repeated words and phrases in a text. It doesn't account for meaning, structure, or synonyms.
  - Word Collocations and Co-occurrences: it identifies words that appear together frequently (bigrams and trigrams) or in the same corpus, aiding understanding in the semantic structure.
  - TF-IDF: Frequency counts the important words in a document within a set of documents. The higher the scoring, the more relevant the words.
  - RAKE (Rapid Automatic Keyword Extraction): Utilizes stopwords and phrase delimiters to identify and then score the most relevant words or phrases in the text.
- Linguistic Approaches: These methods usually rely on linguistically salient cues towards identifying key words and phrases in a text. This consists of information down to the morphology and syntax, including part-of-speech tags and dependency grammar relations, that aides in qualifying the importance of keywords. For instance, nouns and noun phrases are scored higher because of the richness they have towards the conveyance of textual information.
- Machine Learning Approaches: Machine learning is used to power systems for text analysis tasks, such as keyword extraction. These systems convert unstructured text into numerical vectors to capture essential features. Further, techniques like Support Vector Machines (SVM) and deep learning are applied to efficiently extract meaningful keywords from the textual data.

2.10.3 Implementation:

For our case we went with keyBERT, which is a minimalistic keyword extraction technique that leverages BERT embeddings to create keywords and keyphrases that are most similar to a document[27].

The way keyBERT works is very simple but a very effective method for keyword extraction, First, BERT is used to obtain embeddings for the whole document, corresponding to a document-level representation. Second, embeddings for N-gram words or phrases are obtained. Third, cosine similarity finds the words or phrases that bear the most resemblance to the document. These high-similarity words will be used as descriptors that contain the essence of the whole document[27].

2.11 Semantic matching:

2.11.1 Introduction:

Semantic matching in its basic form is about assessing whether two or more elements in our case piece of text data if they have similar meaning, it uses machine learning and neural network to develop a deeper understanding of the language and learn the relationships between the words and their meaning[32].

2.11.2 Semantic matching techniques:

Semantic matching relies heavily on neural networks, these neural networks process large volumes of text data to extract vector representations capturing the meaning of individual words and phrases—word embeddings. These facilitate mathematical comparisons of semantic similarity.

Different neural network architectures for semantic matching have been developed for specific use cases:

- Convolutional Neural Networks (CNNs) are used to classify and process the meaning of sentences for understanding the meaning of phrases.
- Recurrent Neural Networks (RNNs) process sequences of text, such as sentences, to understand context and ordering.
- Transformers capture the meanings between all words in a sentence or document, determining meaning at a deeper level.

2.11.3 Semantic matching and query understanding:

Semantic matching enhances understanding the user's query. the core process in understanding the user's query phase in our system is extracting keywords from the user's query and instead of directly performing syntactic matching semantic matching is employed instead, this will provide much greater flexibility and engance the retrieval of important courses to the user

2.11.4 Semantic matching strategies:

Before we start with discussing the approaches that we gonna follow to implement semantic matching, the question that could be asked here is that why are we going to use neural networks for the task where a basic approach based on topic modeling just similar to the method used to create the dataset for this task, there are two main reasons for this. Firstly these basic method like using topic probability distribution similarity is inherently noisy and since it's probabilistic modeling, thus the dataset created from this method will be inherently noisy, this is where the role of neural networks is important which is the ability to enhance and update their representation of the data, through feedback loop mechanism. Secondly, the goal here is to find and descover methods that can be used to create labeled datasets from unlabeled complex data like textual data that represent natural language.

To train a neural network for semantic similarity, the typical way is to have a labeled dataset for this task where the labels represents similarity scores between two pieces of text, in our case we don't have a labeled dataset for this task, therefore different approaches are considered:

- **Document co-occurence:** if two keywords or keyphrases appear in the same document, they are considered semantically related and they are assigned a label of 1 indicating they semantically related, otherwise if they don't appear in the same document they are consider semantically unrelated and given a a label of 0.
- **Enhanced co-occurence with topic modeling:** the first approach can somehow be effective but it neglects the interconnectedness and the semantic relationships between the documents(courses) to enhance the previous approach topic modeling is used, so in this case if two keywords or keyphrases appear in two different documents instead of considering them as semantically unrelated we compare the the similarity of topic probability distribution of their documents, and to determine if they should be semantically related with do that according to a specific threshold, the challenge though is in determining the optimal threshold and due to the complexity in natural language different threshold for different keywords is needed, so for this a dynamic threshold is employed instead, for each keyword we experiment with different thresholds and to determine the appropriate threshold, we use a coherence measure to determine the appropriate threshold for each keyword, to measure the coherence we use a pretrained word embedding model and fine tune it to our dataset, By averaging the embeddings of keywords considered semantically related at different thresholds, we select the threshold with the highest coherence score for each keyword.
- **Simple constrastive learning for sentence embedding(SIMCSE):** A framework harnessing contrastive learning to generate state-of-the-art sentence embeddings. SimCSE, which stands for Simple Contrastive Learning of Sentence Embeddings, has advanced the state-of-the-art noticeably. SimCSE generates sentence embeddings using both unsupervised and supervised approaches.

  The unsupervised approach is where an input sentence is predicted against itself through contrastive learning with dropout noise. This creates two different embeddings of the same sentence, treated as positive pairs, while other sentences in the mini-batch serve as negative pairs. It can match the performance of previous supervised methods, with dropout acting as minimal data augmentation to improve robustness.

  In the supervised approach, labeled pairs are used from natural language inference (NLI) datasets. Pairs labeled "entailment" serve as positive examples, and those labeled "contradiction" serve as hard negatives. This improves even further the performance of sentence embedding with respect to the unsupervised method. Since we are interested in the unsupervised approach due to the lack of unreliable of labeled dataset. Unsupervised SimCSE optimizes contrastive learning to generate highly effective sentence embeddings. The core idea is to predict an input sentence against itself by using only dropout as noise. The process works as follows:

  Dropout as Data Augmentation: Dropout is a well-known regularization technique in training neural networks. We can view it as data augmentation in a minimal way. Passing a sentence through a pre-trained encoder, dropout makes its mask randomly drop some neuron activations, so the sentence representation is slightly different each time. By applying dropout twice to the same sentence, we get two different embeddings, which we call positive pairs. When fed into the same sentence twice, dropout will lead to different embeddings, which will be known as positive pairs. This process allows the model to learn different views of the same text. In other words, it helps the model be more robust and avoids collapse in representation.

Create Positive Pairs: The two different embeddings of the same sentence, which are created using dropout, are regarded as positive pairs. The positive pairs are very important for the training process so that the model learns to identify similar sentences even when they have some difference.

Use In-Batch Negatives: In addition to the positive pairs, all the other sentences in the same mini-batch also serve as negative pairs. This means, for each sentence, the model should try to identify its positive pair from all other sentences (negatives) in this mini-batch. Unsupervised SIMCSE uses a contrastive loss function which is designed to maximize the similarity between similar instances and minimize the similarity between negative pairs, For a pair of sentence embeddings h_i and h_{i+}, the objective function uses cosine similarity and is defined as:

$$\mathscr{L}_i = -\log \frac{e^{\text{sim}(h_i,h_{i+})/\tau}}{\sum_{j=1}^{N} e^{\text{sim}(h_i,h_{j+})/\tau}} \tag{2.5}$$

where τ is a temperature parameter that controls the scaling of similarities, and N is the batch size[23].

- **Simple contrastive learning for sentence embedding with topic modeling:** In the unsupervised approach SIMCSE uses in-batch negatives as negative pairs, this could consider even pieces of text that are semantically related to address this we can use topic modeling as a guide to choose negative pairs in a more controllable way, based on a threshold we take the topic probability distribution of a sentence or a keyphrase that we want the model to learn its semantic embedding and choose sentences that are basically dissimilar according to the threshold and use it as a negative pair instead.

2.11.5 Model architecture:

The previous proposed approaches require different architectures that will be discussed below:

For the co-occurence and the enhanced co-occurence approach using topic modeling two different architectures can be followed:

1. **Siamese Netowork:** a siamese network consist consists of two identical siamese networks or branches that share the same architecture and weights, hence the name, each branch will process one input of the text as the network takes two inputs that represent the two pieces of text that we want to infer their semantic similarity score. The siamese network can be designed and implemented in two possible way, both sharing the first layers:

   Each sub-network or branch will share the same architecture:

   - **Embedding Layer**: Converts input text into numerical representations.
   - **LSTM or BERT Transformer**: Processes representations of the text, extracting contextual information.
   - **Fully connected**: After the encoder, produces embeddings of the same size for both input texts.

   From here the network can be designed in two different ways:

- In this design each sub-network produces an embedding representation of its input, in this design we use a constrastive loss function with a margin the goal is to maximize the similarity between similar instances, the margin in this loss function represents a boundary that defines the acceptable distances for inputs to be considered as similar.

$$L = \frac{2}{N} \sum_{i=1}^{N} y_i \cdot d_i^2 + (1 - y_i) \cdot \max(0, \text{margin} - d_i^2) \tag{2.6}$$

  where:

  - N is the batch size,
  - y_i is the label indicating whether the pair of inputs are similar (1) or dissimilar (0),
  - d_i is the distance between the embeddings of the pair of inputs,
  - *margin* is a hyperparameter specifying the minimum distance between dissimilar pairs.

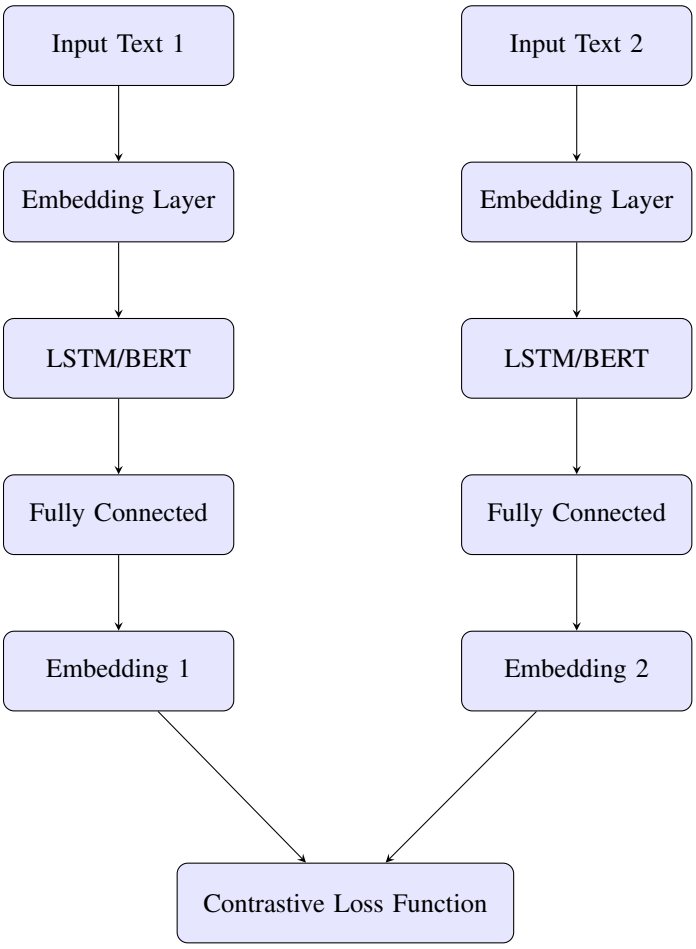


Figure 2.7: Semantic matching: Siamese network with constrastive loss model architecture

One of the main advantages of using this architecture is the interperability since the final output of the network is basically a distance that represent the relationship between the two inputs, this allows for easier and straightforward interpretation of the output.

- The alternative design is that after each subnetwork produces an embedding representation of its input, both embeddings are concatenated and processed by another fully connected layer followed by a sigmoid function that produces a probability score indicating the similarity between the two inputs where a binary cross-entropy can be used as a loss function for this case.

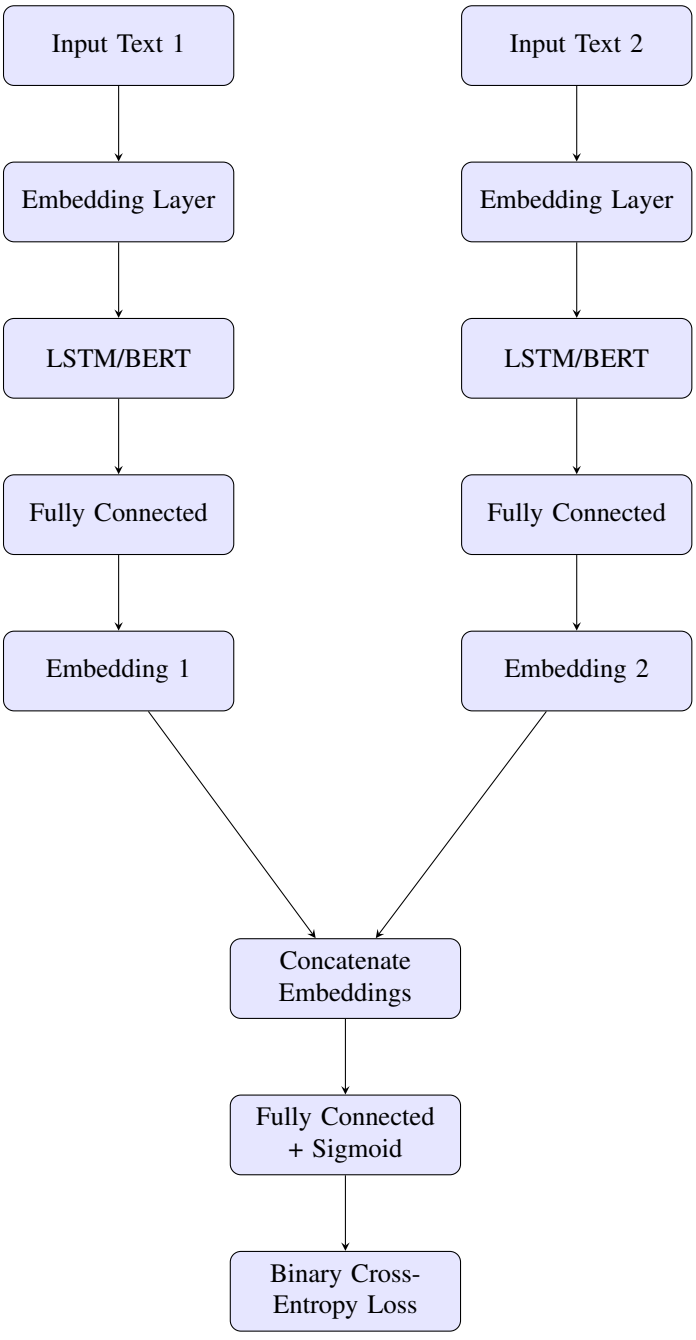


Figure 2.8: Semantic matching: Siamese Network with Binary Cross-Entropy Loss model architecture

2. **Cross encoder:** Across encoder which is a network architecture used mostly in NLP, as it can take multiple inputs by concatenating them into a single representation and the network can

learn the relationship between those inputs according to the task. A cross encoder architecture can be employed as the following:

- **Input representation:** Concatenate the two sequences into a single sequence of inputs. This can either be done through explicit concatenation with a special separator token or through a fixed structure for combining the input.
- **Embedding layer:** This part of the model would embed the concatenated input sequence of tokens into a dense vector.
- **LSTM\BERT:** Forward the concatenated sequence to a BERT model, which captures contextual interactions using its transformer architecture, or to an LSTM network, which captures the interactions through sequential dependencies.
- **Attention mechanism (optional):** It can be used to focus on the parts of the sequence that are more important for the prediction task.
- **Output layer:** This will be a fully connected (dense) layer followed by a sigmoid activation.

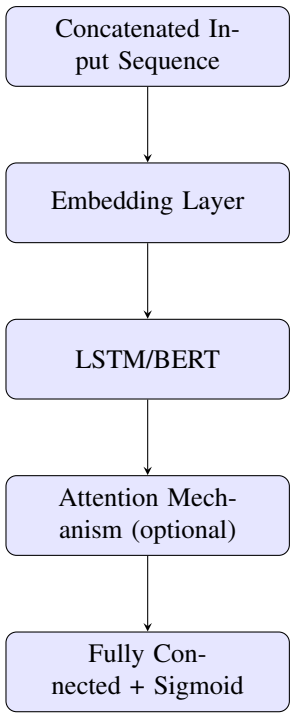


Figure 2.9: Semantic matching: cross encoder model architecture

This archicture

3. **SIMCSE:** For simple contrastive learning of sentence embeddings, we used the architecture defined in the SimCSE paper[23]. We used dropout as a minimal data augmentation to create positive samples while utilizing in-batch negatives as negative pairs. The model uses BERT to generate embeddings for each input. Each input is fed into the BERT model twice; during the second pass, dropout is applied, which leads to a different version of the input. Afterwards,

a feed-forward network maps the positive pairs (a sentence and its augmented version) close together, while pulling the negative samples (in-batch negatives) away from it[23]. The model architecture can be described as follows:

- **BERT Encoder:** generate embeddings for input sentences.
- **Fully Connected Layer:** A fully connected layer follows BERT to generate the final embeddings.

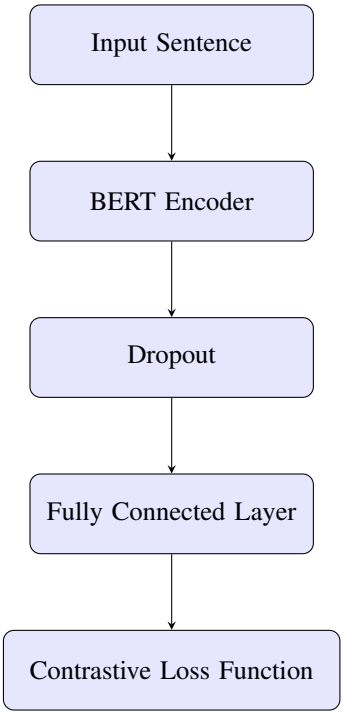


Figure 2.10: Semantic matching: SimCSE model Architecture

2.12 Query expansion

2.12.1 Introduction:

Query expansion is a process applied in information retrieval systems with the objective of refining search results' precision. It involves adding more terms and phrases to what a user inputs in their first search query to increase the scope and relevance of what is to be brought back. This process can be automated, generating new terms from the original query, or it can involve adding synonyms and related terms to the query[3].

2.12.2 Query expansion techniques:

Query expansion may be carried out in several different ways, depending on the information retrieval system and goals of a search. Some of the more common techniques include:

- **Thesaurus-based expansion:** A thesaurus can be used to find synonyms for words from a given document and identify additional terms considered relevant.

- **Co-occurrence expansion:** Identifies terms that frequently occur in documents along with the original query.
- **Ontology-based expansion:** Conceptually similar to the thesaurus-based expansion, but in this case, one uses a structured hierarchy of terms.
- **Pseudo-relevance feedback:** Initial search results are used to identify terms that occur disproportionately highly in the collection, and those terms are added to the query terms. These methods enrich the efficiency and relevance in the results of a search.

These methods enrich the efficiency and relevance in the results of a search[3].

2.12.3 Proposed approaches to query expansion:

For query expansion to consider two potential approaches:

- **LSTM-Based Model**:

  We intend to use an LSTM network in a sequence-to-sequence model. Such models are usually trained on labeled data such that the input (query or keywords/keyphrases from documents) is linked to its extended version. Because labeled data for this task is not available for us, we consider the dataset that was used previously in semantic matching task, based on topic modeling. We take semantically similar associated keywords from that dataset, and that makes the first element of each pair, and the second element is its expanded version. This is in line with the use of labeled data for training according to the passage above about graph neural networks.

- **Graph Neural Network Approach:**

  Upon analysis, we observed that the extracted keywords and keyphrases naturally coalesce into a dense graph, derived directly from the document's content. In this graph, individual words serve as nodes, interconnected by edges representing their co-occurrence relationships. Leveraging this inherent graph structure, we propose employing a graph neural network (GNN) for the task of query expansion.

  The ascendancy of graph neural networks in recent years underscores their profound capability to discern and exploit intricate structural nuances and relational dynamics within graphs, as elucidated in the excerpt provided. Given the paucity of labeled data tailored explicitly for graph-based challenges, our strategy entails training the GNN via unsupervised methods. The crux of this approach lies in acquiring meaningful node embeddings adept at encapsulating the graph's topology and the nuanced relationships among keywords[**?**].

  These learned embeddings hold substantial promise in facilitating link prediction[38], enabling the network to forecast relationships among keywords extracted from the corpus of documents. This predictive capacity amplifies the network's utility, enhancing the efficacy of query expansion by inferring latent connections among terms, thereby enriching the semantic context and scope of search queries.

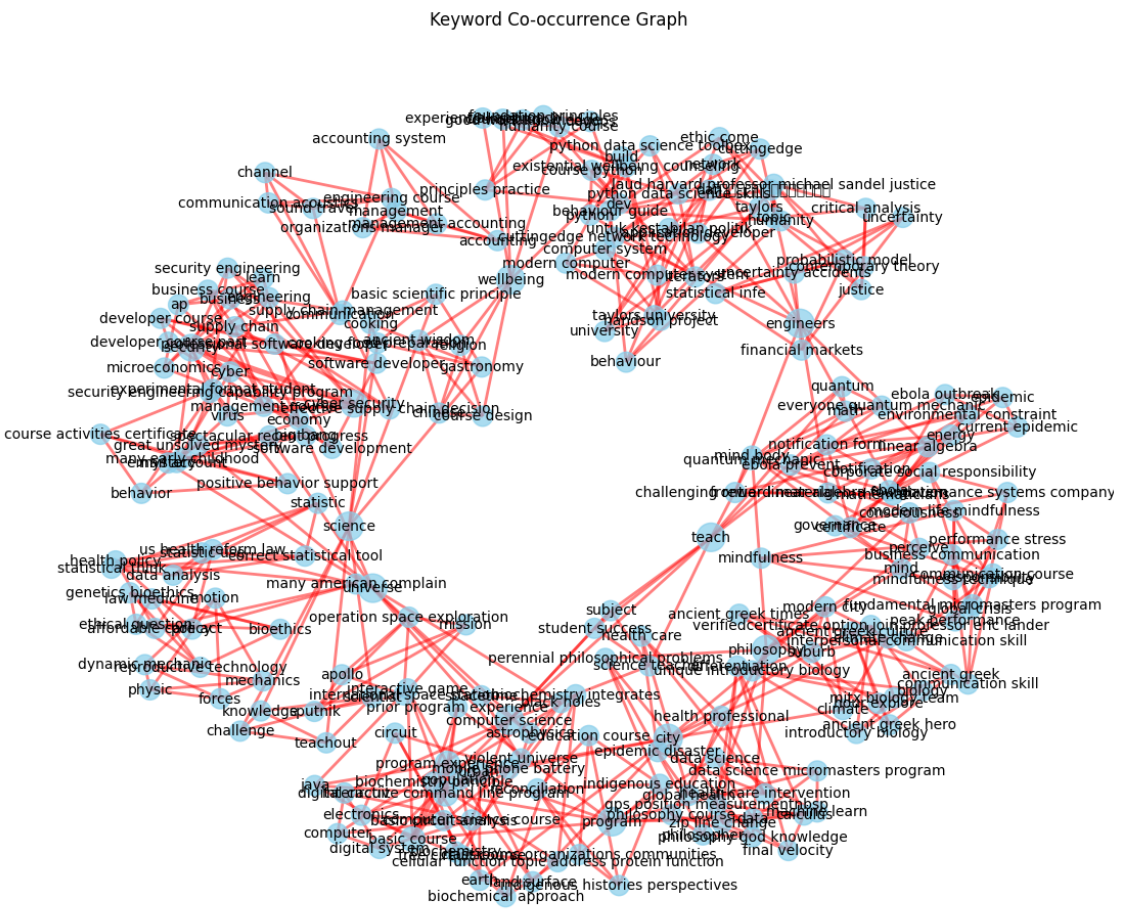


Figure 2.11: Keyword co-occurence graph of 100 keywords

2.12.4 Model architecture:

We propose the following model architectures that can be used to implement the query expansion in our system:

- **Seq2Seq model:** A Sequence-to-Sequence (Seq2Seq) model, also called an Encoder-Decoder model, consists of two recurrent neural networks: the encoder and the decoder. While an input sequence goes into the encoder to get a compressed representation, the representation is utilized by the decoder to produce an output sequence[2].

  For an encoder-decoder architecture with LSTM the following architecture is followed:

  - **Encoder:**
    - **Input Layer**: Receives an input query in the form of a word sequence.
    - **Embedding Layer**: Encodes the input query, word by word.
    - **LSTM Layer**: Processes the sequence of encoded words and outputs a sequence of hidden states and a final hidden state (context vector).

- **Decoder:**
  - **Input Layer**:

    The initial hidden state of the input layer is the final hidden state of the encoder.
  - **Embedding Layer**:

    It converts the input tokens into some dense vector space.
  - **LSTM Layer**:

    One word is generated at a time, considering the previous hidden state and the embedding of the current input token.

    The sequence of hidden states and output at every time step.
  - **Fully Connected Layer**:

    The LSTM output is passed through a fully connected layer to make the size of the output equal to the size of the vocabulary, producing logits for each word in the vocabulary.
  - **Softmax Layer**:

    The logits are passed through the softmax layer to convert them into probabilities for each word in the vocabulary.
  - **Output**:

    The word which is in the vocabulary with the highest probability of occurrence is taken as the output token at that particular time step.

    The same word will serve as an input token for the next time step.

A suitable loss function would categorical cross-entropy loss.

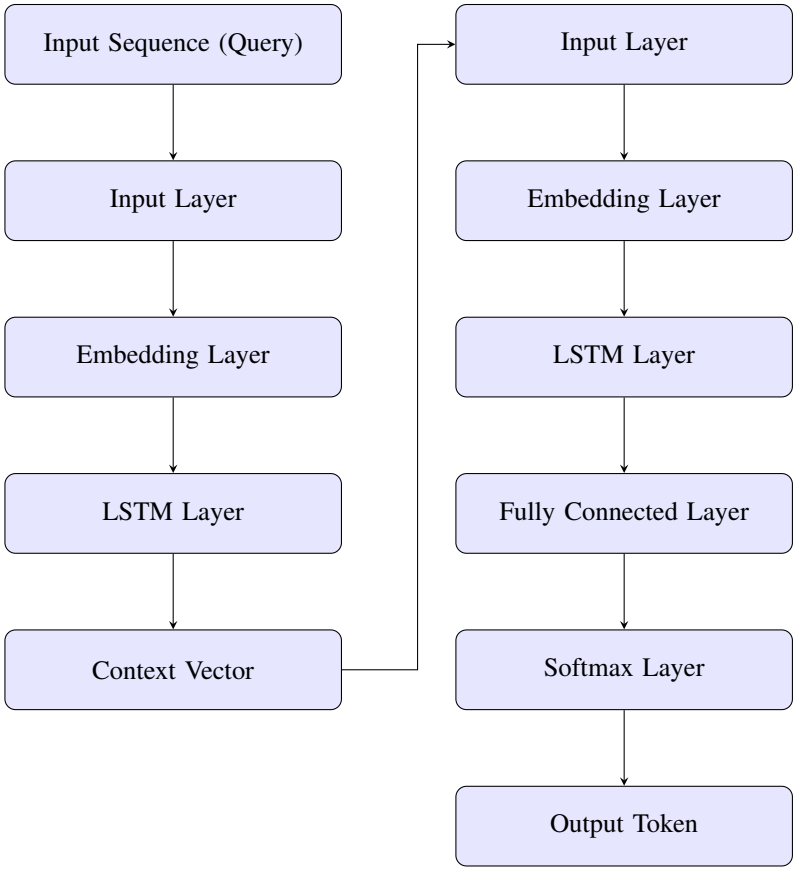


Figure 2.12: Query expansion: Seq2Seq Architecture with LSTM model architecture

- **Graph neural network:** Before going into the actual architecture we need first to decide what type of graph neural network we going with.
  - **Graph Convolutional Networks (GCN)**:
    - GCNs are the most-cited and widely-used GNN architecture. They provide a unified framework for simplifying ChebNets, considering only 1-hop neighborhoods and enforcing self-connections.
    - They symmetrically normalize the graph Laplacian with a renormalization trick to make the training stable.
    - They are both efficient and effective, but edge features are not directly supported.
  - **Message Passing Neural Networks (MPNN)**:
    - MPNN is a model that first proposed the concept of message passing: ”nodes pass the message to each other through the edges.”
    - These messages are computed using features of the nodes and the edges and then aggregated to update the representation of the nodes.
    - MPNNs are highly expressive, but storing and processing all messages on edges can face scalability issues.
  - **Graph Attention Networks (GAT)**:

    - GATs bring attention mechanisms to the graph: learn the importance of the features of neighboring nodes in a dynamic manner.
    - This is attractive because the attention mechanism allows the network to perform well by focusing attention on the most relevant parts of the graph.
    - They are computationally efficient, scale well, and can model edge features.

– **Sampling Methods (e.g., GraphSAGE)**:

    - GraphSAGE is an algorithm to perform graph-based learning.
    - GraphSAGE adds sampling techniques to make the process of large graph learning more efficient.
    - Instead of using all neighboring nodes, the idea is to sample a subset and aggregate their features.
    - This reduces computational load and makes the approach scalable.
    - GraphSAGE can be trained using supervised, unsupervised methods, and many different aggregation functions[30].

From among the array of methods considered, our focus has been narrowed down to two approaches that are very promising: Graph Convolutional Networks and GraphSage, to decide which one to go with, we need to consider two important concepts when it comes to graph neural networks which are inductive and transductive learning:

- **transductive learning:** in this case the model learns from both training and test data upfront, the model sees all the nodes in the graph during training and during testing. However, as a new node is added, the model needs to be retrained[30].
- **inductive learning:** the model looks only at the training data. It then makes predictions of unseen data by only looking at this initial training process[30].

GCN and GraphSAGE follow different learning strategies. While GCN adopts a transductive approach, performing embedding generation for all nodes in the graph at the same time, with the whole graph structure, and using it during the training process, GraphSAGE follows an inductive approach. It could learn embeddings for individual nodes and improve them with the incorporation of new nodes in the graph or a new graph by synthesizing information from its local neighborhoods[9].

For this reason we decided to go with GraphSAGE due to its main strength is inductive learning on graph data. The inductive learning offers generalization to new nodes and graphs, and it improves the scalability of the methods in real-world scenarios characterized by a continuous data influx[9].

What is GraphSAGE: GraphSAGE, expanded as SAmple and aggreGatE, is a breakthrough in generating low-dimensional embeddings for the nodes of large graphs. Unlike its predecessors, the algorithm is designed in an inductive manner, supporting generalization to unseen nodes or even entirely new subgraphs. This feature is crucial in current applications working with dynamic graphs, e.g., social networks or citation graphs, in which new nodes continuously appear[28].

The key innovation of GraphSAGE is its ability to learn representations on nodes through sampling and aggregating features of a node's local neighborhood. Instead of learning an explicit embedding for every unique node, GraphSAGE generates node embeddings by aggregating feature information from a node's local neighborhood using a set of aggregator functions.

GraphSAGE uses three kinds of aggregators to aggregate the features of node neighbors: Mean Aggregator, LSTM Aggregator, and Pooling Aggregator. Mean Aggregator takes the mean value of feature vectors from its neighbors, which is similar to convolutional propagation in GCNs. LSTM Aggregator uses LSTM networks to model the order information of neighbor feature vectors, hence enhancing its expressiveness. Pooling Aggregator aggregates neighbor feature vectors through fully connected layers and performs element-wise max-pooling, thus allowing varied neighborhood feature aggregation types.

To use graphSAGE for our task we use this general architecture:

1. **Input Layer**: Takes in the node features and adjacency matrix.
2. **Aggregator Layers**: Graph Convolutional Layer to aggregate neighbor information and refine node representations.
3. **Dense Layer**: Acts as a bottleneck layer that compresses learned representations into a lower-dimensional space.

The unsupervised GraphSAGE model utilizes a loss function specifically designed to encourage similarity between neighbours while forcing more dissimilarity between distant nodes[28]. This is given as follows:

$$J_G(z_u) = -\log(\sigma(z_u^T z_v)) - Q \cdot E_{v_n \sim P_n(v)} \left[\log(\sigma(-z_u^T z_{v_n}))\right],$$

where:

* $J_G(z_u)$ is the loss function,
* σ is the sigmoid function,
* z_u and z_v are the embeddings of the target node u and the context node v, respectively,
* Q is the number of negative samples,
* $P_n(v)$ is a negative sampling distribution, and
* v_n are negative context nodes sampled from $P_n(v)$.

But for our case we are going to use another unsupervised approach more tailored to our task. We use an encoder-decoder architecture where the encoder is basically a GraphSAGE network that produces node embeddings, and the encoder takes these embeddings as input and tries to reconstruct the graph by link prediction and the loss function here is the reconstruction loss we can use binary-cross entropy where the decoder produces 1 if they two nodes are connected otherwise 0 if they are not.

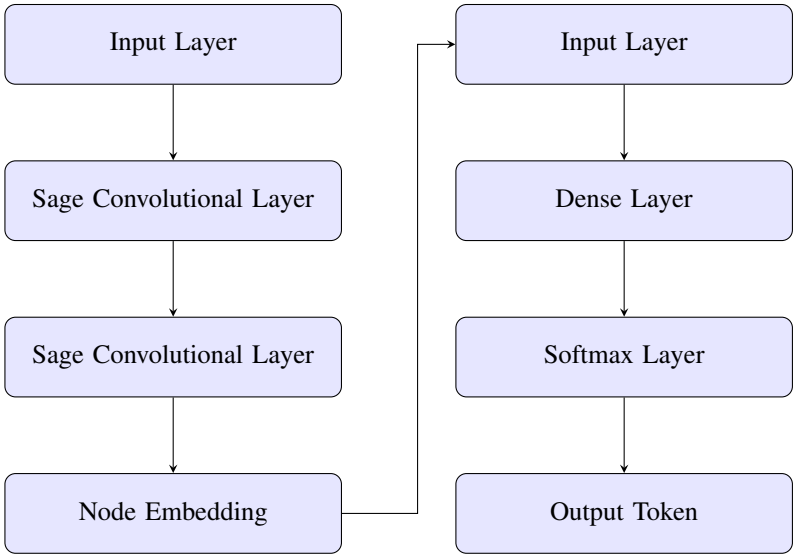


Figure 2.13: Query expansion: Seq2Seq Architecture with Sage Convolutional Encoder and Dense Decoder

2.13 Relevance evaluation

2.13.1 Introduction

Our systems is composed of multiple components as the goal was to create a modular search engine system, despite that one of the most important components of the system is the relevance model where it takes the chosen courses from the previous components and produce the final relevance value for the user.

2.13.2 Proposed approach

In our case we propose BERT, as a siamese neural network to rank and evaluate documents based on their relevance to the given query in the system. In the initial phase of the system and due to the lack of labeled data and the using a synthetic data in this important part of the system, instead we use a variant of the BERT model that is optimized for semantic similarity tasks. However as the system interact with users we can then enhance the BERT-based siamese network model using this feedback loop.

2.13.3 Model architecture

As we described in the approach that we are going to use a siamese network where its sub-networks is basically BERT where it can be followed by a dense layer to enhance the model's even more, this followed by a sigmoid function that produces the relevance score.

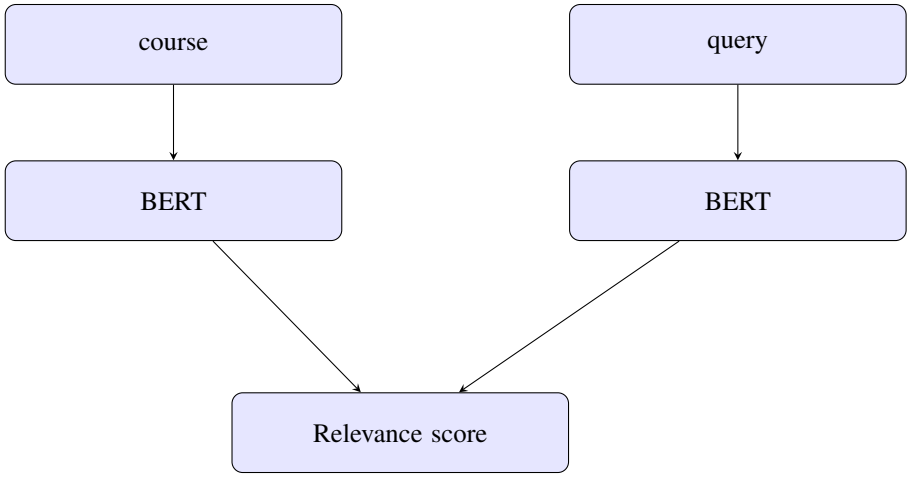


Figure 2.14: Semantic matching: Siamese network with contrastive loss model architecture (revised)

2.14 Feedback loop

One of the important components in any system specifically and most importantly machine learning based systems and which our MOOC platform depends on to improve its components based on the user's needs which is the feedback loop which is basically a mechanism used to continually update and refine the system and its representation to match the user's needs, upon the user receiving their search results they are encourage to provide a feedback on the search results and even each individual course in the results.

There are two types of feedback loops:

- **Positive feedback loop:** this is where the user indicates that the search results provided are relevant, this tells the system that its overall output is valid and its behavior gets encourage in the case where the confidence of the systems is low in its results.
- **Negative feedback loop:** this happens when the user indicates that the search results or some courses are not relevant to his query, this cause the system to undergoes several changes and updates:
  - **Semantic matching model:** the keywords extracted from the user's query and the keywords from the courses or their textual descriptions based on the user's feedback are used to update the model's representation to indicate they are not semantically related, this is very important specifically when we choose our semantic matching models instead of the SIMCSE, as we emphasized before there's noise and wrong labeling in the method we used to generate the labeling of the dataset, this feedback loop will correct these errors based on the user's feedback as that was the main goal of using those models.
  - **Query expansion models:** the same thing for the query expansion models where keywords extracted from the courses and the user's query would be used to update the model's representation and indicate that those keywords are not related and the courses they occurs in are not relevant to the user's query.

– **Relevance evaluation model:** the final relevance evaluation model's representation is updated in a way that it will understand that the given user's query and the course that it predicted its relevant is not actually the case and corrects its behavior based on that.

This iterative mechanism continuously adapts to the user's needs by using positive and negative feedback, and ensuring high quality results and improve the search experiene of the user.

2.15 Conclusion

In conclusion, this chapter has thoroughly explored various aspects of semantic matching and query expansion, crucial components in augmenting the efficiency and relevance of information retrieval systems. Semantic matching techniques, including Recurrent Neural Networks (RNNs) and Transformers, were highlighted for their pivotal role in discerning the underlying semantics and context of textual data. These methods empower the system to gauge the semantic similarity between different pieces of text, thereby enhancing query comprehension and search result retrieval, which are particularly pertinent in the context of Massive Open Online Courses (MOOCs).

Additionally, the chapter delved into the practical implementation of semantic matching through diverse strategies, such as document co-occurrence, enhanced co-occurrence with topic modeling, and Simple Contrastive Learning for Sentence Embedding (SIMCSE). These methodologies harness neural networks to glean semantic representations of textual data, thus refining the accuracy of similarity assessments, which is crucial for delivering relevant course materials and resources in MOOC platforms.

Moreover, the exploration extended to query expansion techniques aimed at fine-tuning search results' precision by augmenting the original query with additional terms and phrases, a critical aspect in enhancing the learning experience within MOOCs. Methods such as thesaurus-based expansion, co-occurrence expansion, ontology-based expansion, and pseudo-relevance feedback were elucidated for their capacity to enrich search queries and amplify retrieval effectiveness in MOOC environments.

To operationalize query expansion, the chapter proposed two potential approaches: an LSTM-based model and a Graph Neural Network (GNN) approach, specifically GraphSAGE. While the LSTM-based model adopts sequence-to-sequence architecture, GraphSAGE leverages inductive learning on graph data to generate low-dimensional embeddings for nodes in large graphs, offering promising avenues for enhancing course search and recommendation systems within MOOC platforms.

In summary, the chapter has provided profound insights into the indispensable role of semantic matching and query expansion in information retrieval systems within the context of MOOCs, furnishing a roadmap for the effective implementation of these techniques to bolster search efficiency and user satisfaction. Through the adoption of advanced neural network architectures and graph-based approaches, MOOC platforms stand poised to better decipher user queries and furnish more precise and pertinent search results, ultimately enriching the learning experiences of millions of online learners worldwide.

Chapter 3

Implementation and results

3.1 Introduction

In this chapter, we delve into the intricacies of implementing the advanced search system within the context of Massive Open Online Courses (MOOCs). This phase encompasses the comprehensive development, integration, and rigorous testing of each system component. We meticulously evaluate various methodologies and compare their performance to ascertain the most effective approaches for achieving optimal search results tailored to the unique requirements of MOOC platforms. Furthermore, we explore the tools and frameworks utilized throughout the implementation journey, elucidating their pivotal roles and contributions to the system's development.

Throughout this chapter, we lay out the implementation process step by step, addressing the challenges encountered along the way and highlighting the outcomes achieved. By doing so, we aim to provide a holistic view of the development process and assess the efficacy of the system specifically within the dynamic landscape of MOOCs.

3.2 Development tools and resources

3.2.1 Hardware environment

The above-described search engine has been developed and tested on the CPU and GPU resources of the cloud with its required capacity to perform the described procedures. A DigitalOcean Linux server was utilized for all CPU-bound activities, such as preprocessing the data, model evaluation, and logical operations.

- **CPU Processing Environment**

  A DigitalOcean-provided Linux server was used to do CPU-intensive tasks like Data Preprocessing, Model Evaluation, Logical Operations.

- **GPU Processing Environment:** Kaggle notebooks were used for tasks that required major GPU computation, especially in training neural networks.

3.2.2 Software resources

Despite the fact that different hardware environments were used, the development and testing processes converged in the GNU/Linux operating system, which contributed to the overall consistency and compatibility of the software at any stage of development. Software resources were used:

Specification	Details
Provider	DigitalOcean
CPU	Intel(R) Xeon(R) Platinum 8280 CPU @ 2.70GHz (4 cores)
RAM	8GB
Storage	100GB SSD
Operating System	Linux

Table 3.1: Server Specifications

Specification	Details
GPU	NVIDIA Tesla P100 (or equivalent)
RAM	13GB
Storage	Cloud-based but with access to Kaggle datasets
Development Environment	Pre-configured with essential data science libraries

Table 3.2: Kaggle Notebook Specifications

- **Python:** Python is a free and open-source, high-level programming language with dynamic semantics and high-level syntax. Being an interpreted object-oriented language, it excels in its readability, modular design, and extensive standard library. Python, due to the existence of built-in data structures and dynamic typing, is extremely well-suited for rapid application development and as a scripting language. Other developers find that its readability and comprehensive standard libraries, which are based on modularity, help to improve code reuse. It is hence loved by developers for its productivity features, such as quick edit-test-debug cycles, and ease of debugging. Python is a favorite for the simplicity and introspective power it possesses with respect to proficient software development[12].
- **Vim:** Vim has been the shining beacon of efficient text manipulation since the early days of UNIX and, more recently, OS X from Apple. It is often included under the name of v̈ifor both platforms. With its rock-stable performance and continual development, Vim adds multi-level undo tree persistence, a rich plugin system, and strong support for a huge number of programming languages and document file types, search and replace, and hooks into virtually every utility available the operating system[15]. Neovim improves on Vim and builds on its strong foundation: combining the features of Vim with modern enhancements that make it an even more powerful and efficient text editor. Neovim is rooted in a commitment to improve the Vim experience through features that improve usability, with a lot of helpful features and improvements that add to the familiar Vim. Rooted in the Vim tradition, Neovim is designed for users looking for the best aspects of Vim combined with modern enhancements to provide an efficient and versatile editing environment[10].
- **Keras:** Keras is a powerful building block for contemporary machine learning infrastructures. It is recognized for its ease of use and general applicability by both industry leaders and research scientists. Keras abstracts away complexity with a simple, consistent API, so it is straightforward to use in training, prototyping, or production deep learning models. Its integration with TensorFlow and other ecosystems makes it a powerful ingredient in the toolkit of any developer[31].
- **Numpy:** NumPy is the fundamental package of Python for scientific computing with an array object and a powerful collection of functions for fast array operations. At the base level

is the ndarray object, allowing you to work with very fast and natively compiled code for n-dimensional array handling that has a uniform data type. For this reason, NumPy is an important package for a wide range of different scientific and mathematical applications because its compiled code allows the fast execution of mathematical operations and the handling of arrays. Its wide use attests to it across many fields, making it a basic tool for data manipulation and analysis in Python[11].

- **Scikit-learn:** usually shortened to sklearn, is one widely adopted, open-source machine learning framework custom-made for Python users. It provides a wide variety of both supervised and unsupervised learning methods and covers tasks like classification, regression, clustering, and dimensionality reduction. Built on top of fundamental Python libraries like NumPy, SciPy, and Matplotlib, it guarantees a user-friendly integration approach into the world of data science. The user-friendly, uniform API makes its use simpler and increases its compatibility, which is one reason practitioners find it time-saving and perfect for machine learning tasks[36].

3.3 Implementation and Experimentation: Evaluating Approaches and Demonstrating Results

In this section, we dwell on the practical implementation of the various components that make up our advanced search system. Each element is developed and integrated very critically, leading to the life of the system. We present these implementations here in the results section, providing detailed expositions and discussions to bring to light the effectiveness of our methodologies. Through analysis of these results, we understand the implications and contributions to the improvement of the search experience. This exploration offers a comprehensive view of the system development process with a spotlight on the practical steps taken and the outcomes achieved.

3.3.1 Topic modeling:

When it comes to implementing BERTopic, in order for the model to run effectively, consideration for several parameters is important, Provided below are the core components and their settings, along with elaborate descriptions:

- **Document embedding:** BERT embeddings transform textual data into numerical representations capturing contextual information and semantic relationships to enhance the model's discriminative ability for topics. For our case we use "all-MiniLM-L6-v2" model.
- **Dimensionality reduction:** We use the default dimensionality reduction option for BERTopic due to its ability to its effective preservation of the local and global structure of the embeddings in their original embedding space. UMAP has different parameters that need to be considered:
  - **Number of Components:** The number of dimensions in the reduced space. A smaller number of components will reduce computational burden while attempting to maintain the critical structure of the data.
  - **Minimum Distance:** This parameter controls the minimum point-to-point distance in the low-dimensional space. This parameter impacts the point spread or the density of the clusters, thus impacting the global layout and structure of the topic clusters.

  - **Number of Neighbors:** This controls the local neighborhood size used for manifold approximation. With a higher value, the point will get a more global structure of the manifold; with a lower value, it focuses more on the local structures.

- **Clustering algorithm:** As BERTopic allow us to choose different clustering algorithms, for our case we'll just go with HDBSCAN due to its ability to define clusters for data with irregular distribution. HDBSCAN works according to different parameters that we can tune to get the best results:

  - **Minimum Cluster Size:** The minimal count of points that should form a cluster. This helps to filter out noise and ensures only significant clusters are identified, thus making the topics more robust and relevant.

  - **Minimum Samples:** The number of samples in a neighborhood for a point to be considered as a core point. It is a key parameter that will help in controlling density and granularity of the clusters, thus building the sensitivity of the model to noise and outliers.

- **BERTopic Parameters:** Since BERTopic uses the previous components to produce the final topics, for that BERTopic has its own parameters to control and shape the output of the previous components to produce the final results:

  - **Number of Topics (k):**
    The number of topics to be identified. This is a crucial parameter to set properly, as it must ensure that the model does not capture too much of the base structure of the data while not over-fragmenting or oversimplifying the topics.

  - **Top N Keywords:**
    An integer value that gives the number of top keywords which should be associated with each topic. The parameter helps to identify the most typical words associated with the topic. This parameter helps in identifying the most representative words for each topic, aiding in topic interpretation and characterization.

To determine the optimal parameters of BERTopic, a grid search approach was employed, whereby a range of values for each of the parameters was systematically explored. It was aimed to find the combination of parameters giving maximal topic coherence, a commonly used quality metric for topic models. Considering the exhaustive nature of grid search, it was necessary to select a range of values for each parameter, guided by the documentation for thorough coverage of potential values. This is different from random search, which proceeds by randomly selecting values for the parameters based on specified distributions. In this way, a grid search can identify the best possible configuration through a comprehensive exploration of a predefined grid of parameter values.

To optimize BERTopic's performance within hardware constraints, three main parameters were identified based on documentation as significant factors affecting its results. These parameters are:

- **min_cluster_size :** The minimum size of cluster represented.
- **nr_topics:** The desired number of identified topics.

- **n_neighbors:** The number of neighbors used in the local neighborhood size of manifold approximation.

And these are the top 10 results based on the coherence score:

min_cluster_size	nr_topics	n_neighbors	coherence_score
30.0	5.0	50.0	0.9911888849393669
100.0	10.0	30.0	0.9860170288017062
30.0	nan	10.0	0.983179336118342
50.0	nan	50.0	0.9746787643538756
30.0	nan	70.0	0.9703534475278768
50.0	nan	30.0	0.9572009460727716
50.0	5.0	30.0	0.9372237060246446
70.0	5.0	50.0	0.935114574371906
10.0	15.0	10.0	0.9259348119570197
70.0	nan	100.0	0.8984553239880885

Figure 3.1: Topic modeling results

Indeed, the conventional approach to the topic modeling results optimization procedure is limited to optimizing one measurement, such as topic coherence. The parameters are tuned based only on achieving the maximal result for that chosen metric. This, however, neglects the broader context of machine learning evaluation and the possible implications of blind optimization of a single metric.

The book *Artificial Intelligence: A Modern Approach* also tries to reiterate the same point: one should have a holistic approach to the evaluation of machine learning. The book emphasizes not to go blindly optimizing the metrics, considering the wider implications and pitfalls. They hold that such a focus on the optimization of metrics might go into futility, leading to suboptimal results and not really capturing the difficulties of practical applications.

With this objective in mind, one possible approach is using the highest scored parameters as a guideline to condense and improve the results. This can be done by analyzing not only the coherence scores but also the word distributions of the topics and using a variety of visualization methods.

We start by examining the intertopic distance map and topic word distributions for the parameter set with the highest coherence score.

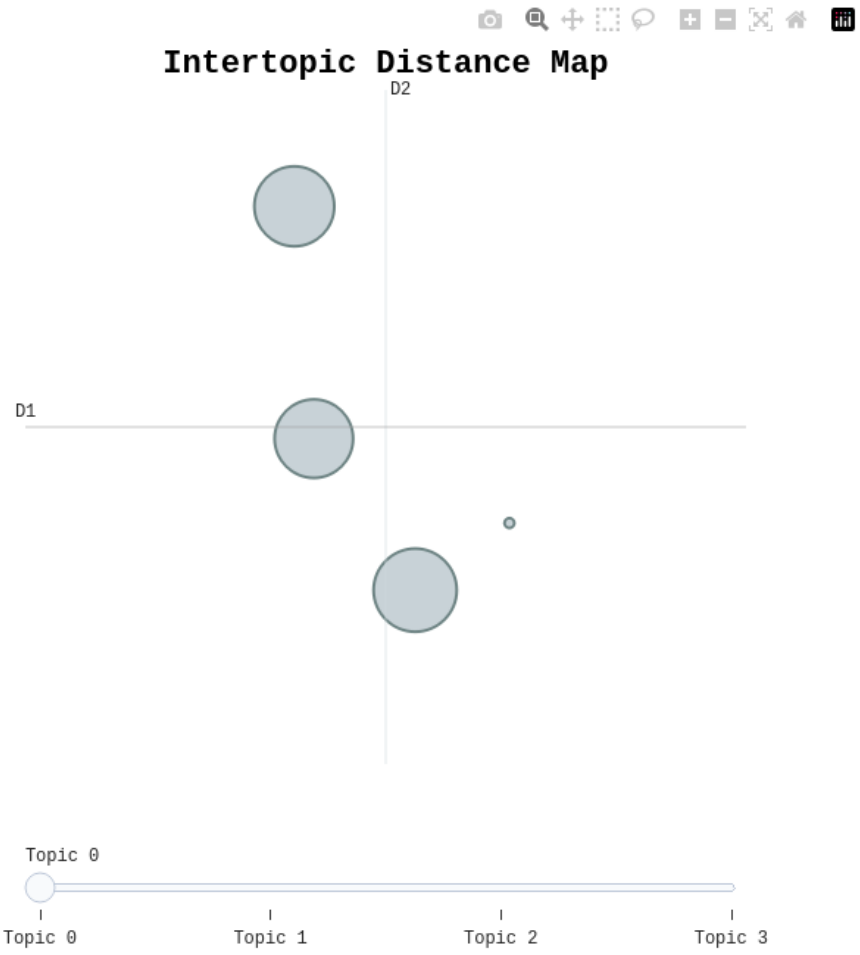


Figure 3.2: Topic modeling: intertopic distance 1

Even though we fix the number of topics at 5, the model found 4 well-separated topics, which is a sign of a sensible separation and consistency of the identified topics.

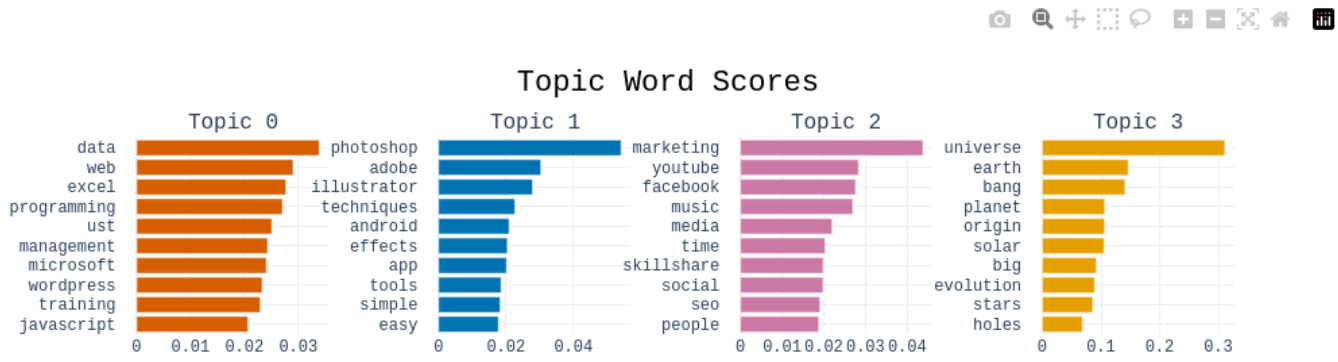


Figure 3.3: Topic modeling: topic-word distribution

In the first topic, we find such coherent words as *programming*, *Microsoft*, and *technology*, though the presence of ”management” is an indicator of some noise or overlap with other topics. The second topic is largely design software with words like *Adobe*, *Photoshop*, *Illustrator*, and *effects*, but brings in some inconsistency with words like ”Android.” The fourth topic seems to be on social media marketing with words like *marketing*, *Facebook*, and *YouTube*, but words like ”time,” ”Skillshare,” and ”people” bring some ambiguity. The final topic is undeniably about space, as we see words like *universe*, *solar*, *holes*, and *stars* that clearly focus on astronomical terms.parameters The 96-topics second parameter set is very representative of overlap and hierarchy in the topics. This representation allows detailed observation of the nuances in the data, which can, however, overcomplicate interpretation. It is quite useful for the exploration of subtopics but requires careful management to yield sensible results.

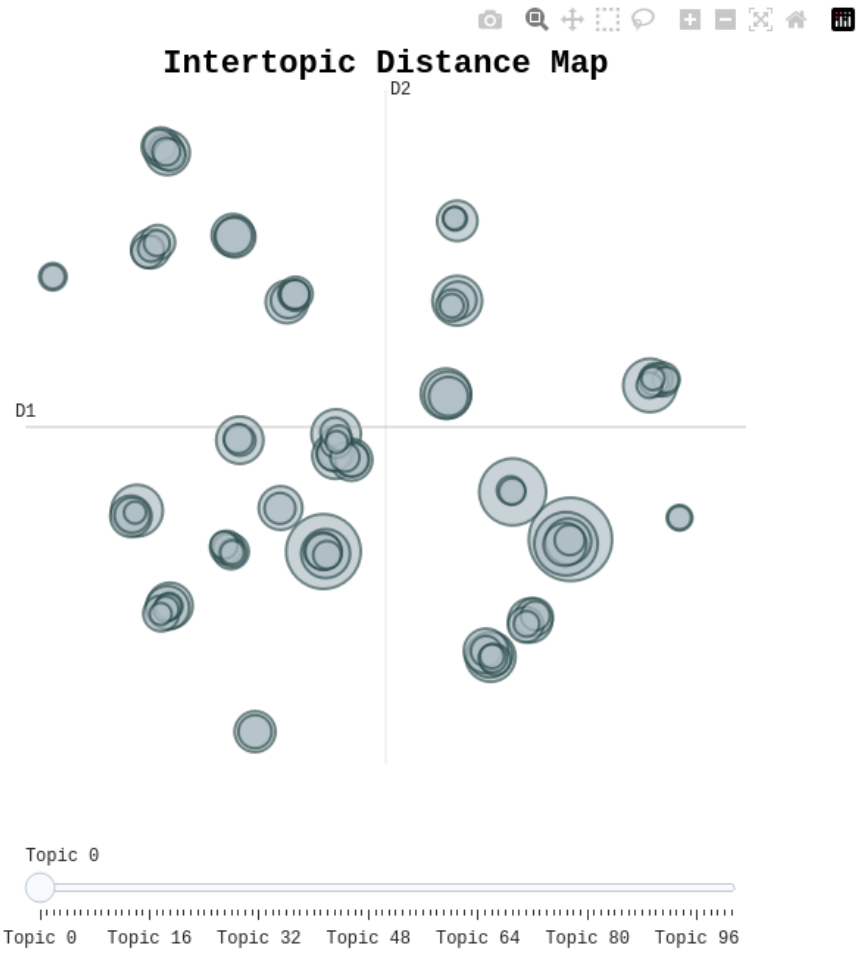


Figure 3.4: Topic modeling: Intertopic distance 2

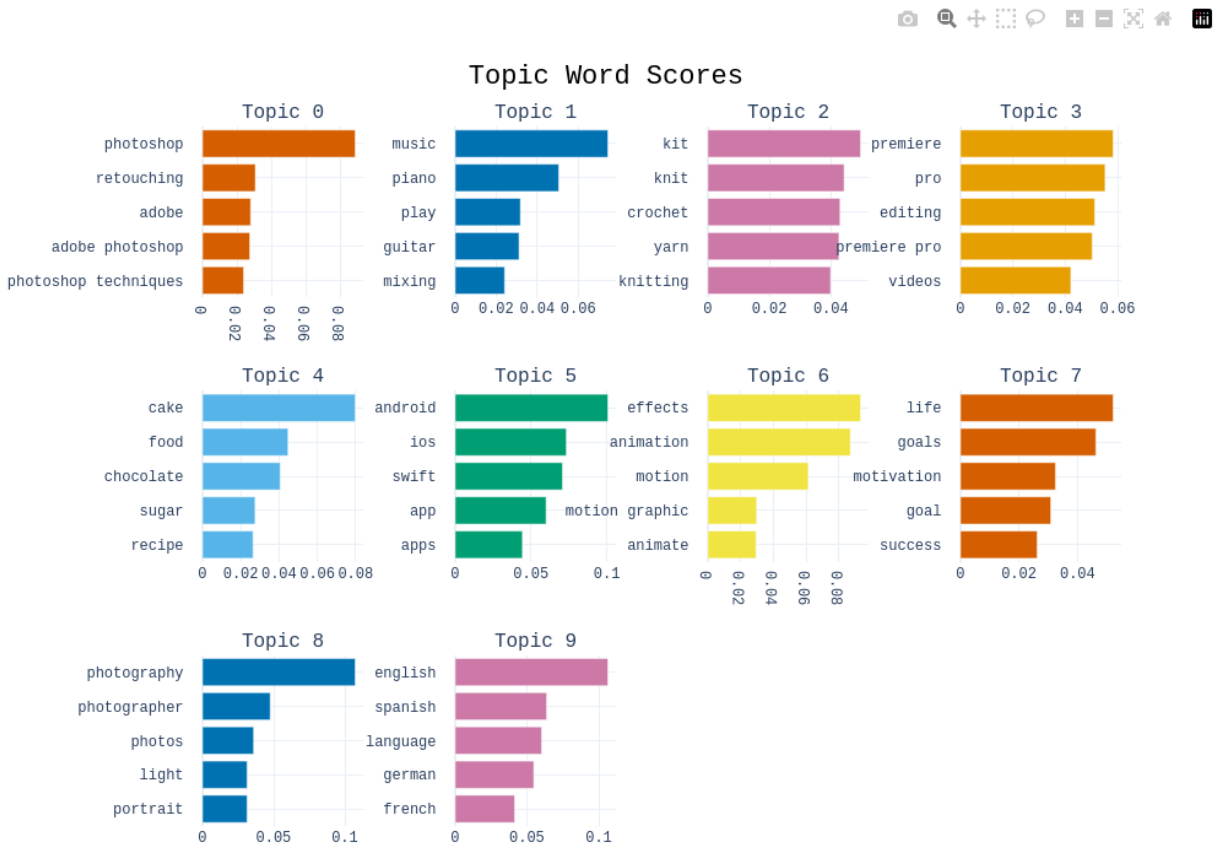


Figure 3.5: Topic modeling: topic-word distribution

- By analyzing the topic word distribution for the first 10 topics, we obtain coherent topics such as graphic design, music, knitting, crochet, yarn, and cooking. But at the same time, certain redundancy and overlapping of these topics are quite visible, which means that probably some topics could encapsulate very similar concepts.

Given the varying results, we can consider using one or both parameter sets for further fine-tuning. Using the first set, we can increase the topics by reducing the parameter `min_cluster_size`, increasing the granularity of the topics. For the second set, we can reduce the topics further by modulating the `nr_topics` parameter to define the number of topics we require and, in addition, increase the `min_cluster_size` to reduce overlap and make more differentiated topics. These adjustments make the topic modeling process flexible in a way that will better serve the desired output in terms of making the topics more coherent and interpretable.

3.4 Text clustering

The purpose of the text clustering section is to classify courses in the dataset, thus providing some structure and insight into the dataset, helping to understand the nature of the data and finally assist

in the identification of bias. Moreover, it improves course retrieval efficiency in the system where the course volume is very large, and it helps in determining the relevant subset for user queries, hence improving system performance.

Each document will be represented by a vector corresponding to topics discovered in the previous section. The goal is to identify groupings of documents that are similar regarding their topic probability distributions or textual data. This clustering will use Jensen-Shannon divergence as the similarity metric in order to group documents that exhibit similar topic distributions or textual content.

Since we have have chosen for DBSCAN as our clustering algorithm, chosen based on the data distribution, it is important to note that it involves two key parameters:

1. **Epsilon (ε):** This parameter defines the radius within which neighboring points are considered part of the same cluster. Points within ε distance are considered core points.
2. **MinPts:** This parameter specifies the minimum number of points required to form a dense region. Points with at least MinPts neighbors within ε distance are considered core points.

The choice of clustering algorithm being DBSCAN, we have to be really careful with the metric we choose for evaluation. Popular metrics like the silhouette score do not take into account the noise in density-based clustering; they are distance measures. This can be misleading because DBSCAN does not only use distances.

As explained in the article[22], traditional metrics such as the silhouette score are not suitable for density-based techniques for clustering due to ignoring the concept of noise and being dependent on distances. Instead, the Density-Based Clustering Validation (DBCV) metric is more relevant. Unfortunately there's no official reliable implementation of this metric, so we are going to show the clustering results using silhouette score and other similar metrics.

epsilon	minpts	silhouette	davies_bouldin	calinski_harabasz	n_clusters	
0	0.100000	2	0.286807	0.873349	2085.538231	35
1	0.100000	3	0.471550	0.750535	2277.923073	32
2	0.100000	4	0.506055	0.719792	2283.269577	32
3	0.100000	5	0.533032	0.631249	2357.730542	31
4	0.100000	6	0.532114	0.603655	2360.106897	31
...	...	...	...	...	...	...
259	0.687755	5	0.339571	0.570460	78.486935	2
260	0.687755	6	0.339571	0.570460	78.486935	2
261	0.687755	7	0.339571	0.570460	78.486935	2
262	0.687755	8	0.339571	0.570460	78.486935	2
263	0.687755	9	0.339571	0.570460	78.486935	2

Using the parameters with the highest scores in all the metrics we visualize the clusters

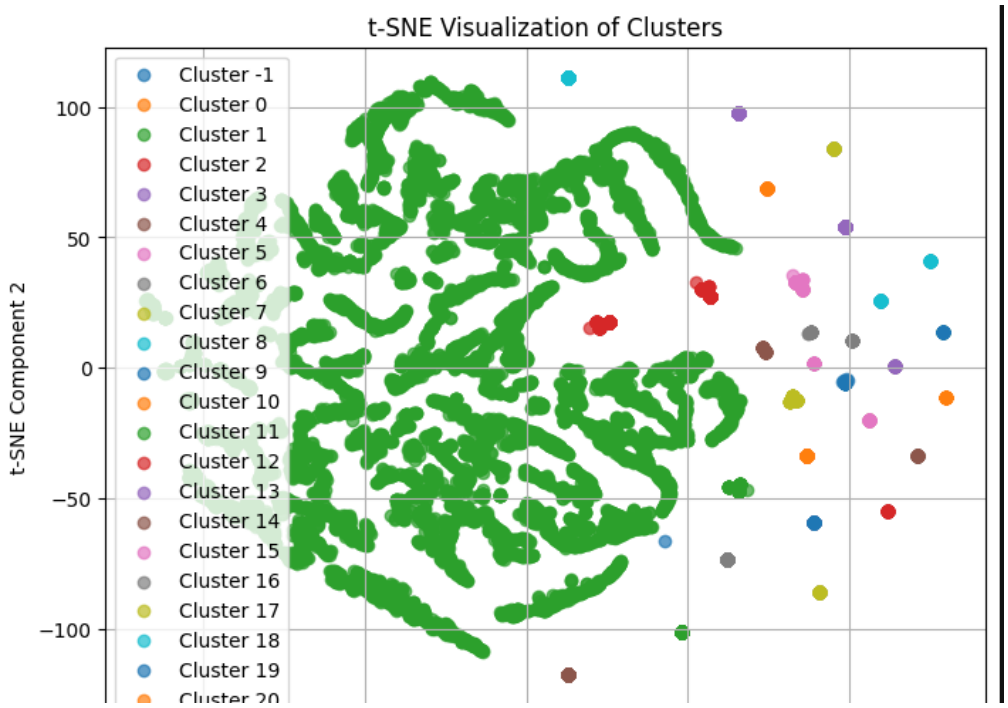


Figure 3.6: Text clustering results

3.5 Named entity Recognition and disambiguation:

This section supplies the implementation details of the Named Entity Recognition and Disambiguation system, by defining the specific approaches presented in the conceptual design section. Each approach is tackled individually to detail its design, implementation details, and methodology. Besides, each method has its data preparation emphasized under the implementation procedure.

3.5.1 Named entity disambiguation

The disambiguation is a critical part in the Named Entity Recognition and Disambiguation system, which links the entities with their probables/actuals in the knowledge graph, like Wikidata. It leverages the Wikidata API to access the knowledge graph and retrieve relevant information on entities and their relations. This information further enhances richness in the dataset and builds upon the knowledge base of the system.

	name	entity_id	doc_idx	topic_similarity	part_of	instance_of
0	turn	Q1365530	1335	0.26726617941771463	[{'id': 'Q287618', 'label': 'sequential game',...	[{'id': 'Q151885', 'label': 'concept', 'descri...
1	mindset	Q1339824	5231	0.3777771223890778	[{'id': 'Q77468620', 'label': 'psychology term...	[{'id': 'Q1127759', 'label': 'cognitive bias',...
2	contract	Q188966	123	0.35844948987486486	None	[{'id': 'Q1188693', 'label': 'mind sport', 'de...
3	winrt	Q8024449	11829	0.2404955500518448	[{'id': 'Q108871754', 'label': 'Windows App SD...	[{'id': 'Q756637', 'label': 'application frame...
4	java	Q2063	51	0.6742800227104347	None	[{'id': 'Q235557', 'label': 'file format', 'de...
5	european	Q1286	2140	0.22800549164394637	[{'id': 'Q847514', 'label': 'Alpide belt', 'de...	[{'id': 'Q46831', 'label': 'mountain range', '...
6	english	Q328	2820	0.26373325308119633	None	[{'id': 'Q10876391', 'label': 'Wikipedia langu...
7	computer science	Q111849006	11	0.7947966142300544	None	[{'id': 'Q5633421', 'label': 'scientific journ...
8	data science	Q110262588	182	0.6519990441244335	None	[{'id': 'Q110256698', 'label': 'Stack Exchange...
9	accounting	Q2990807	329	0.5928450030177558	None	[{'id': 'Q309100', 'label': 'planning', 'descr...
...	...	...	...	...	...	...
104296	susan	Q61141616	1495	0.25965282424920194	None	[{'id': 'Q15773317', 'label': 'television char...
104297	android	Q124454023	10907	0.7840145363082069	None	[{'id': 'Q131436', 'label': 'board game', 'des...
104298	minutes	Q107424229	9669	0.7333994264248294	None	[{'id': 'Q7889', 'label': 'video game', 'descr...
104299	embroidery	Q64575001	1306	0.7577235507533642	None	[{'id': 'Q93184', 'label': 'drawing', 'descrip...
104300	sony vegas	Q7562555	5039	0.25823754693901885	[{'id': 'Q28969703', 'label': 'Vegas Pro', 'de...	[{'id': 'Q1373429', 'label': 'video editing so...
104301	english	Q15924427	355	0.4562312947455431	None	[{'id': 'Q101352', 'label': 'family name', 'de...
104302	fractal	Q124038897	1160	0.716636013358382	None	[{'id': 'Q7889', 'label': 'video game', 'descr...
104303	microsoft office	Q11266	477	0.2817425540052651	[{'id': 'Q11255', 'label': 'Microsoft Office',...	[{'id': 'Q11320567', 'label': 'application', '...
104304	wei	Q842157	1244	0.7662167774470661	None	[{'id': 'Q101352', 'label': 'family name', 'de...
104305	every day	Q5417466	7138	0.26216173841535567	[{'id': 'Q7617036', 'label': 'Still Feels Good...	[{'id': 'Q134556', 'label': 'single', 'descrip...

Figure 3.7: Named entity disambiguation results

After the results were fetched from Wikidata, the following step is determining the right entity out of those results based on measures identified in the conceptual design stage, The next step would be to determine how we are going to use those measures.

After examining the reuslts we found that the semantic similarit measure by taking the cosine similarity of the embeddings produced by the BERT model gave better results and for that we are going to rely on it to determin the correct entity from the possible candidates.

	name	entity_id	topic_similarity	semantic_similarity	instance_of
45808	python	Q271218	0.17849661372766376	0.27200260758399963	[{'id': 'Q16521', 'label': 'taxon', 'description': 'group of one or more organism(s), which a ta...
46059	python	Q28865	0.2934731883896692	0.4273972511291504	[{'id': 'Q899523', 'label': 'object-based language', 'description': 'programming language'}, {'i...
...	...	...	...	...	...
90640	python	Q76417859	0.20594577972976533	0.045708976686000824	[{'id': 'Q101352', 'label': 'family name', 'description': 'part of a naming scheme for individua...
90644	python	Q3411448	0.22283657206402974	0.34652817249298096	[{'id': 'Q506240', 'label': 'television film', 'description': 'film format that is broadcast and...

Figure 3.8: Named entity disambiguation results on the word python

We can deduce from the results that among entities with identical names, the entity most closely related to the entity `python` presented the maximum cosine similarity in its embedding produced by BERT.

3.5.2 Named entity recognition

In the Named Entity Recognition subsection, we used named entity disambiguation-related data from the previous subsection to train our model. It involved the extraction of the entity with the highest similarity score and the extraction of its instances using the relationship We then used these

associated entities' names as classes or labels of our dataset, and to help the model focus or put more attention on the named entity as we mentioned in the previous chapter we create an entity mask that helps the models knows the position of the entity in the text. We used Keras to build the model, as its user-friendly nature is particularly convenient when implementing custom models is not required. Keras often becomes a lightweight, efficient, and simple tool for prototyping neural networks.

3.5.3 Representation Learning

Model1: Embed-100-72-LSTM-128-72-EntityAttention-Dense-50

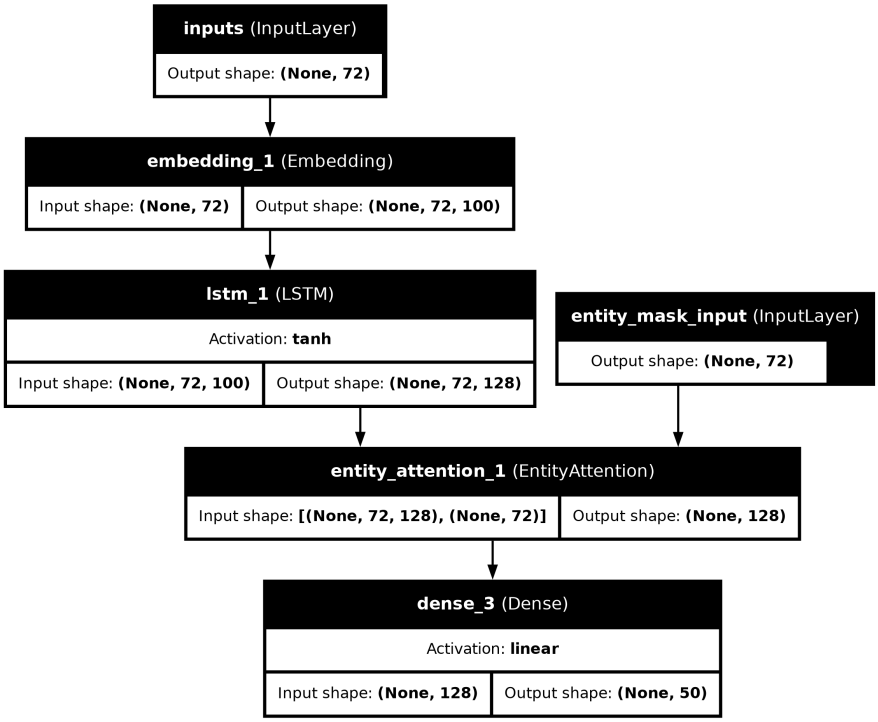


Figure 3.9: Named entity recognition: Representation learning model implementation

The model starts with an embedding layer that transforms the input sequences(tokens) into 100-dimensional vectors, followed by that is an LSTM layer with 128 units that processes the input sequence to capture sequential patterns over time, where return_sequence is set to True to make the LSTM produces an output for each timestep. A custom entity mechanism comes next where attention scores for the named entities are increased according to the entity mask provided to the network as a second input, this will enable the model to put more attention on the target entity in the sequence. Finally the model utilizes a Dense layer to produce the final 50-dimensional embedding. this model didn't involve any non-linear activation functions, the only exception is the LSTM layer which uses the hyperbolic tangent function(tanh) for its job.=

Figure 3.10: NER(repesentation learning): Model 1 loss chart

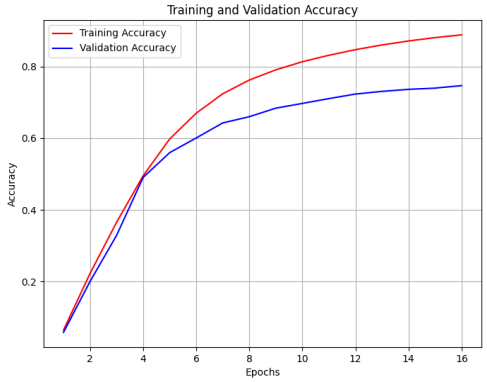


Figure 3.11: NER(Representation learning): model 1 accuracy chart

Figure 3.12: Named entity recognition Loss and accuracies charts for Model1

The model stopped at the epoch 16 due to early stopping. the model starts with an initial training accuracy of 6.5% and a training loss of 1.10. As for the validation the model started with an initial validation accuracy of 5.9%, and with a validation loss of 1.22 despite this low initial performance. However, as the training progresses through successive epochs, both training and validation accuracies increase, similarly the validation and training loss continues to decrease , achieving a training accuracy of 88.8% and a validation accuracy of 74.7% in the final epoch. Correspondingly, the training loss reduces to 0.29, and the validation loss decreases to 0.68, despite that we can see a significant gap between the training and validation both loss and accuracy at the final epochs indicating that the model is suffering from overfitting.

Dropout 0.4:

Figure 3.13: NER(Representation learning): Model 1 loss chart with dropout of 0.4

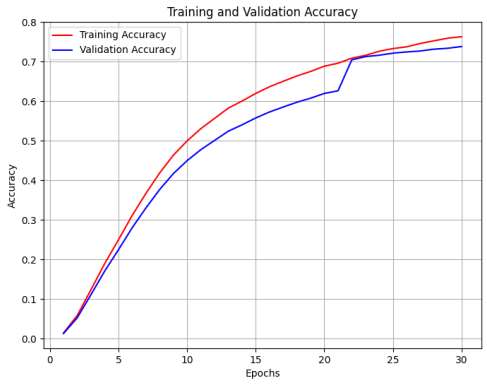


Figure 3.14: NER(Representation learning): Model1 for accuracy chart with dropout of 0.4

Figure 3.15: NER(Representation learning): loss and accuracy charts for model 1 with dropout of 0.4

applying a dropout of 0.4 to the base model showed signficant impact on the performance of the

model both training and validation. the training accuracy started with a lower value of 0.0148, while the validation accuracy started with 0.0133. Similarly, the model started with an initial training loss of 1.1438 and validation loss of 1.2582. The model reached a final training accuracy achieved was 0.7633, and the validation accuracy was 0.7386. The corresponding final training and validation losses were 0.5255 and 0.6312.

3.6 Semantic matching:

For semantic matching, we implemented three approaches: a Siamese network with two main different architectures using, a cross-encoder approach, and the SimCSE approach. Each approach has its own data preparation and format requirements each tailored to its specific needs.

3.6.1 Siamese network

Constrastive loss

Both approaches uses a labeled data created using a topic modeling as a guidance, both approaches have the same data requirements.

- **Model 1: Embedding-LSTM64-LSTM64-Dense64-DistanceCalculation-ContrastiveLoss**

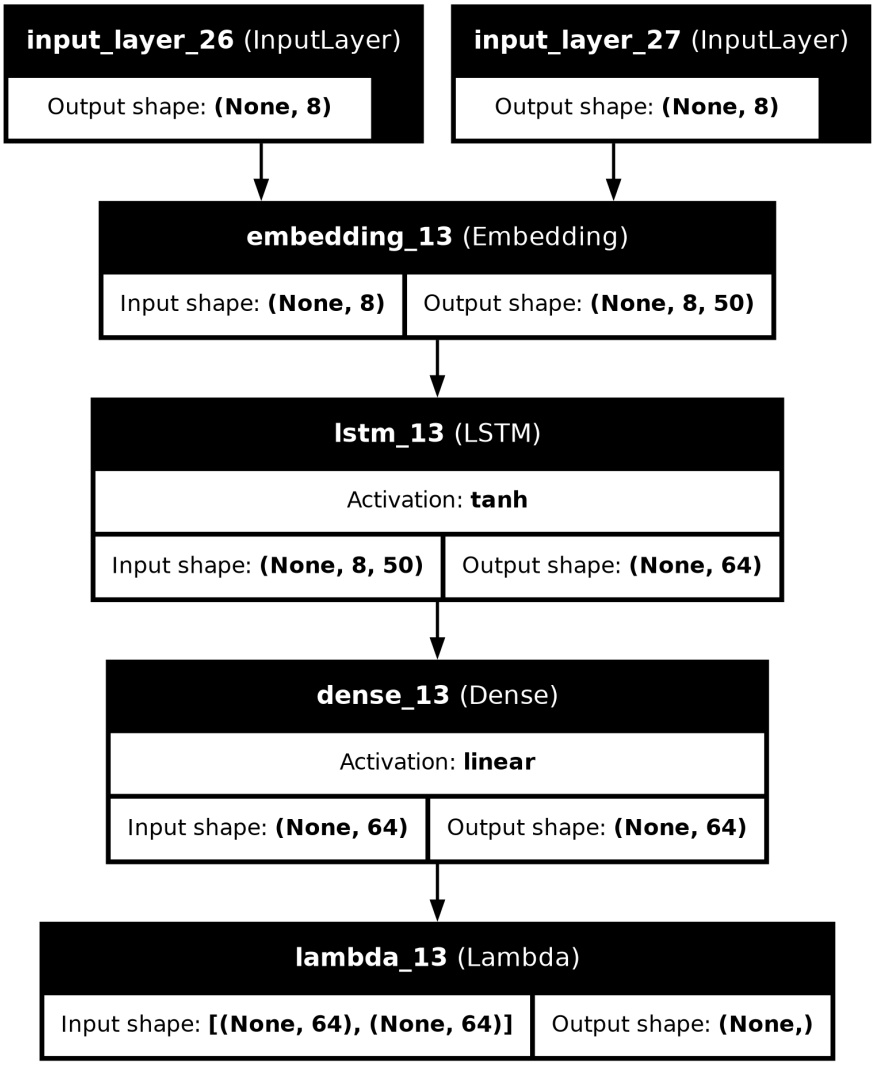


Figure 3.16: Semantic matching: Constrastive loss model architecture

This Siamese network contains two identical subnetworks with shared weights. Each subnetwork takes one of the two inputs and is composed of the same architecture: an embedding layer with a shape of (8, 50) transforms each sequence into dense vector representations of size (8, 50). Then these embeddings are fed to an LSTM layer with 64 units. Following the LSTM layer, a dense layer produces a 64-dimensional embedding for each subnetwork. The outputs generated from both subnetworks are then concatenated into a single vector of shape 128. After that , a lambda layer calculates the Euclidean distance between the embeddings, which is then fed into the contrastive loss function. The final prediction of semantic similarity relies on the distance between the embeddings of the two inputs. If the distance falls within the predefined distance treshold which is usually the margin, indicating semantic relatedness, then the prediction is 1; otherwise, it's 0 for semantic unrelatedness.

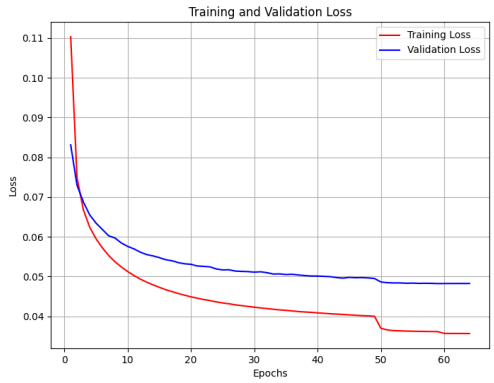


Figure 3.17: Semantic matching(Constrastive loss): model 1 loss chart

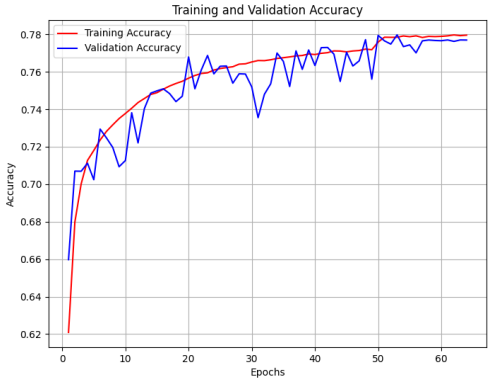


Figure 3.18: Semantic matching(Constrastive loss): model1 accuracy chart

Figure 3.19: Semantic matching(Constrastive loss): loss and accuracy charts for model 1

The model was stopped at 64 epoch due to early stopping triggered to the lack of improvement in the validation loss. The initial train loss was 0.1459, accuracy 58.85%, whereas validation loss was 0.0831, with an accuracy of 65.98%. The model continued to improve in terms of training loss as well as the validation loss during training. Finally, it ended up having a training loss of 0.0356 and a validation loss of 0.0483. The training accuracy went to 78.01%, and the validation accuracy stabilized around 77.70%. An interesting behavior can be seen approximately around the epoch 50 where the loss exhibits a sharp decrease for both the loss and validation this can be due to the adjustment of the learning rate during training which helped the model to improve faster, After epoch 50 we can see that the loss and accuracy for both training and validation started to stabilize with no significant improvements. Furthermore the large gap between the training and validation loss and the fluctuation of the validation accuracy indicate signs of overfitting. Overall the model showed improvements in both training and validation accuracy.

- **dropout 0.1:**

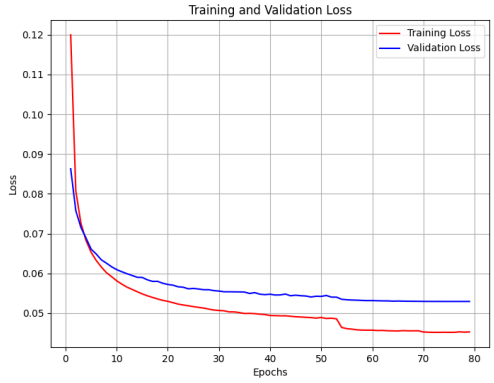


Figure 3.20: Semantic matching(Constrastive loss): model 1 loss chart with dropout 0.1

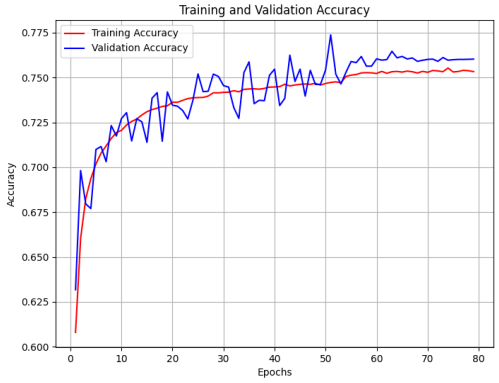


Figure 3.21: Semantic matching(Constrastive loss): model 1 accuracy chart with dropout 0.1

Figure 3.22: Semantic matching(Constrastive loss): model 1 accuracy and loss charts with dropout 0.1

By comparing the the results of the base model to the results of adding dropout of 0.1 we can analyze the effects of the dropout on the model and the training process, the model with dropout started with a lower training accuracy of 57.77% with a slightly lower validation accuracy at 63.18% compared to the base model, that started with training accuracy of 58.85% and validation accuracy of 65.98%. the initiation training loss was 0.1565 for the dropout model, which was higher than that for the model without dropout at 0.1459. The initiation validation loss was 0.0863 for the dropout model, slightly lower than that of 0.0831 for the model without dropout. adding the dropout made the model exhibit a gradual and consistent improvements with the progress of the accuracy and loss for both the training and validation, The model reached 75.54% and 76.03% for training and validation accuracy, by epoch 79, which is lower then the base model. despite that the model showed reduced fluctuations when it comes to the validation accuracy and reduced the gap between the validation and training loss as well the accuracy which means the dropout helped in reducing overfitting, the model stopped at 79 epochs due to early stopping compared to the base model which showed faster convergence.

The margin in the constrastive loss function represent the boundary that separates similar embeddings from disimillar embeddings in the embedding space, decreasing the margin in this will enforce the embeddings to be closer together in the embedding space thereby enhancing the model's discrimination

Margin 0.5

The model with a margin of 0.5 started with an initial training accuracy of 57.96% and initial validation accuracy of 58.85%. decreasing the margin to 0.5 showed faster convergence, achieving its highest training accuracy of 76.88% by epoch 42 and a peak validation accuracy of 77.85% by epoch 39. in addition to that the validation loss decreased more consistently, reaching a low of 0.0122 compared to the base model's minimum of 0.0512. despite not achieving accuracy level as the base model with a margin of 1, based on results reducing

the margin to 0.5 enhanced the model's convergence stopping at the epoch 48 due to early stopping compared to the base model who stopped at 68 epochs.

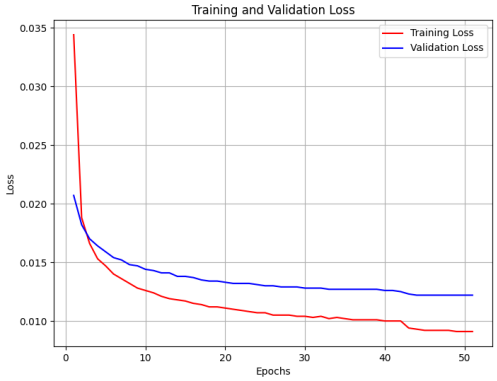


Figure 3.23: Semantic matching: loss chart for model 1 with margin 0.5

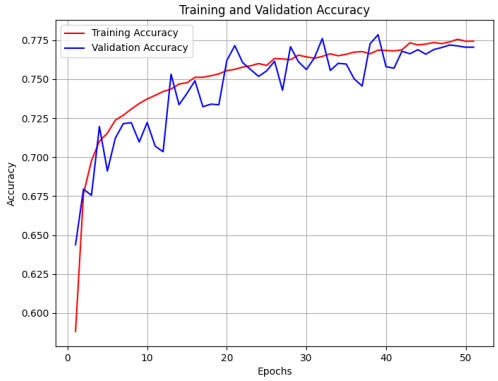


Figure 3.24: emantic matching: accuracy chart for model 1 with margin 0.5

Figure 3.25: emantic matching: loss and accuracy charts for model 1 with margin 0.5

margin 2:

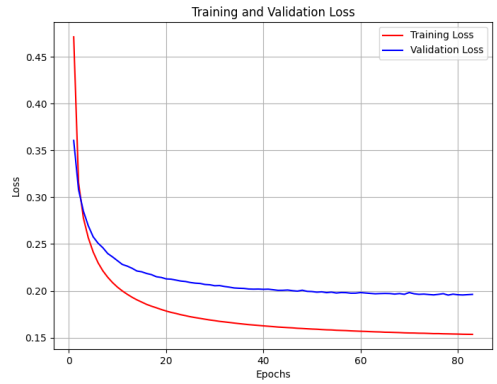


Figure 3.26: emantic matching: loss chart for model 1 with margin 2

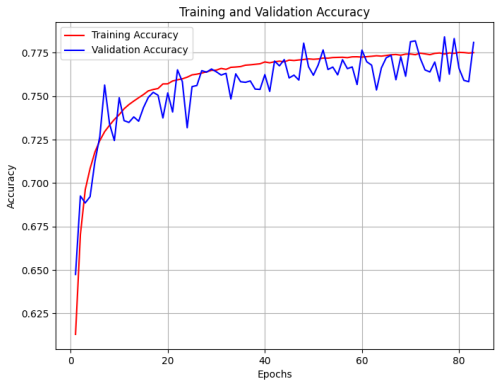


Figure 3.27: Semantic matching: accuracy chart for model 1 with margin 2

Figure 3.28: emantic matching: loss and accuracy charts for model 1 with margin 0.5

After increasing the margin to 2, the model showed different behavior but achieving similar performance when it comes to the the loss and accuracy, The base model had an initial 56.24% training accuracy with 0.6538 loss and 55.86% validation accuracy starting with 0.6481 loss. The model with a margin of 2 started with an initial 57.49% training accuracy and a training loss of 0.6472. For the validation accuracy, the model with the increased margin started at

56.45% with a loss of 0.6394. The increased margin model had a final training accuracy of 77.24% and loss of 0.1551 with a validation accuracy of 75.68% and a loss of 0.1962. The base model obtained a final training accuracy of 78.01% along with a loss of 0.0356 and a validation accuracy of 77.70% along with a loss of 0.0483 over 64 epochs. not only that the model didn't show better performance but the increased margin model took more epochs to converge, that is, by increasing the margin, the model made the convergence slower despite the comparable accuracies ultimately achieved.

- **Model 2: Embedding-LSTM128-Dense64-DistanceCalculation-ContrastiveLoss** In this model the architecture is the same the only difference is that we increased the size or the number of the LSTM units to provide the model with increased capacity to learn more patterns about the data potentially leading to improved performance.

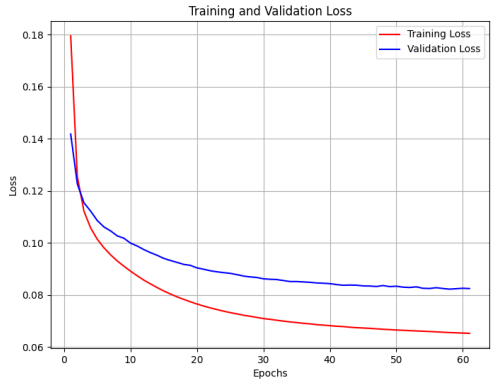


Figure 3.29: Semantic matching: loss chart for model 2

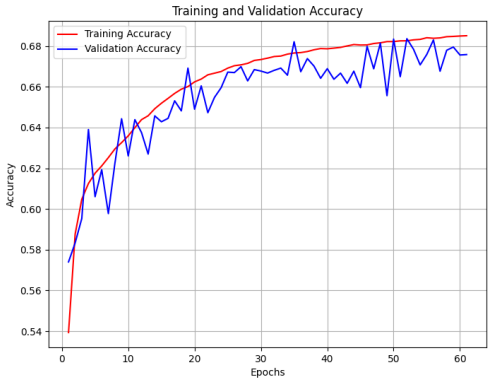


Figure 3.30: Semantic matching: accuracy chart for model 2

Figure 3.31: Semantic matching: loss and accuracy charts for model 2

Since we increased the size of the LSTM units from 64 units to 128 units, it's important to analyze if this increase is going to achieve better results. The model begins with a slightly lower accuracy of 52.06% and a loss of 0.2142 compared to the 64-unit model. However, it showed faster improvement, by achieving a training 67.25% accuracy and a loss of 0.0710 by the epoch 26 , with validation accuracy increasing from 57.40% to 66.72% and the loss decreasing from 0.1418 to 0.0883 compared to the first model which achieved a slightly better accuracy of 68% only after 65 epochs. Overall, the 128-unit LSTM model showed improved and faster convergence compared to the 64-units LSTM model indicating that there's potential for more improvements.

- **Model3:Embedding-LSTM64-Dense32-Dense-32-DistanceCalculation:**

  In the previous architectures we can see clearly the model is struggling even when it comes to the training accuracy. this can be attributed to either the model not having enough capacity to fit the data specifically the training data or due to the noise in the data resulted from using the probabilistic nature of topic modeling and complexity of natural language that this approach can't capture.

For this we add another dense layer to the previous architecture with a shape of (32,).

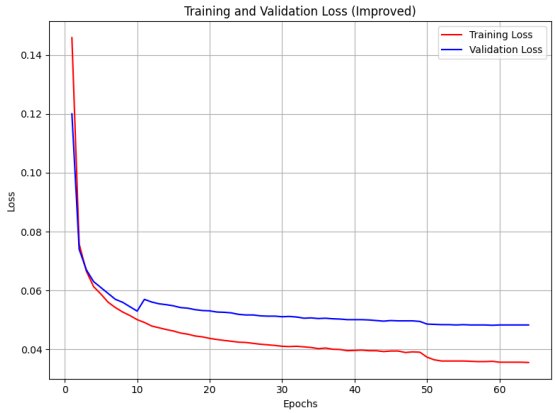


Figure 3.32: Semantic matching(Constrastive learning): loss chart for model 3

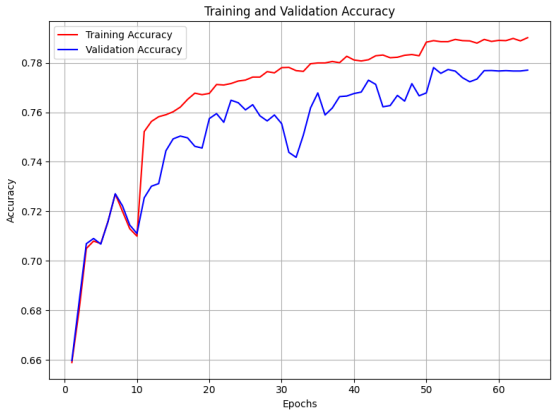


Figure 3.33: Semantic matching(Constrastive learning): accuracy chart for model 3

Figure 3.34: Semantic matching(Constrastive learning): loss and accuracy charts for model 3

The model stopped at the epoch 62 due to early stopping indicating the lack of improvement in validation loss. The model started with an initial training accuracy 0.659 and a validation accuracy of 0.6598, the model showed good performance when it comes to both training and validation accuracy till it reached epoch 8 where both accuracies training and validation started to decrease after reaching a peak of 73% within that range, to a lower accuracy value of 71% at the 10th epoch, the training accuracy started to improve again after that along side the validation aaccuracy reaching a final value of 79% for the training accuracy and a validation accuracy of 77%, the loss showed a similar trend starting with an initial training loss 0.152 and a validation loss of 1.12, both of them showing similar improvement. by the epoch 10, the validation loss started to show slow improvement compared to the training loss reaching a final value of 0.03554 for the training loss and a validation and validation loss of 0.0482.

The gap between the validation and training accuracy as well as the validation and training loss, alongside the validation accuracy fluctuation shows signs of overfitting.

3.6.2

Shared Dense layer

- Model 1:Embedding-LSTM128-Dense64-Dense-64-DistanceCalculation

We follow the same previous architecture of the siamese network with the constrastive loss function, the only difference is that we have added a dense layer at the end that concatenate the outputs of the sub-networks this means this dense layer will have a shape of (128,) since the output of each subnetwork has a shape of (64,), this layer then is followed with a sigmoid function that squeezes the output to a range between 0 and 1 which represent the probability of the two inputs being semantically related.

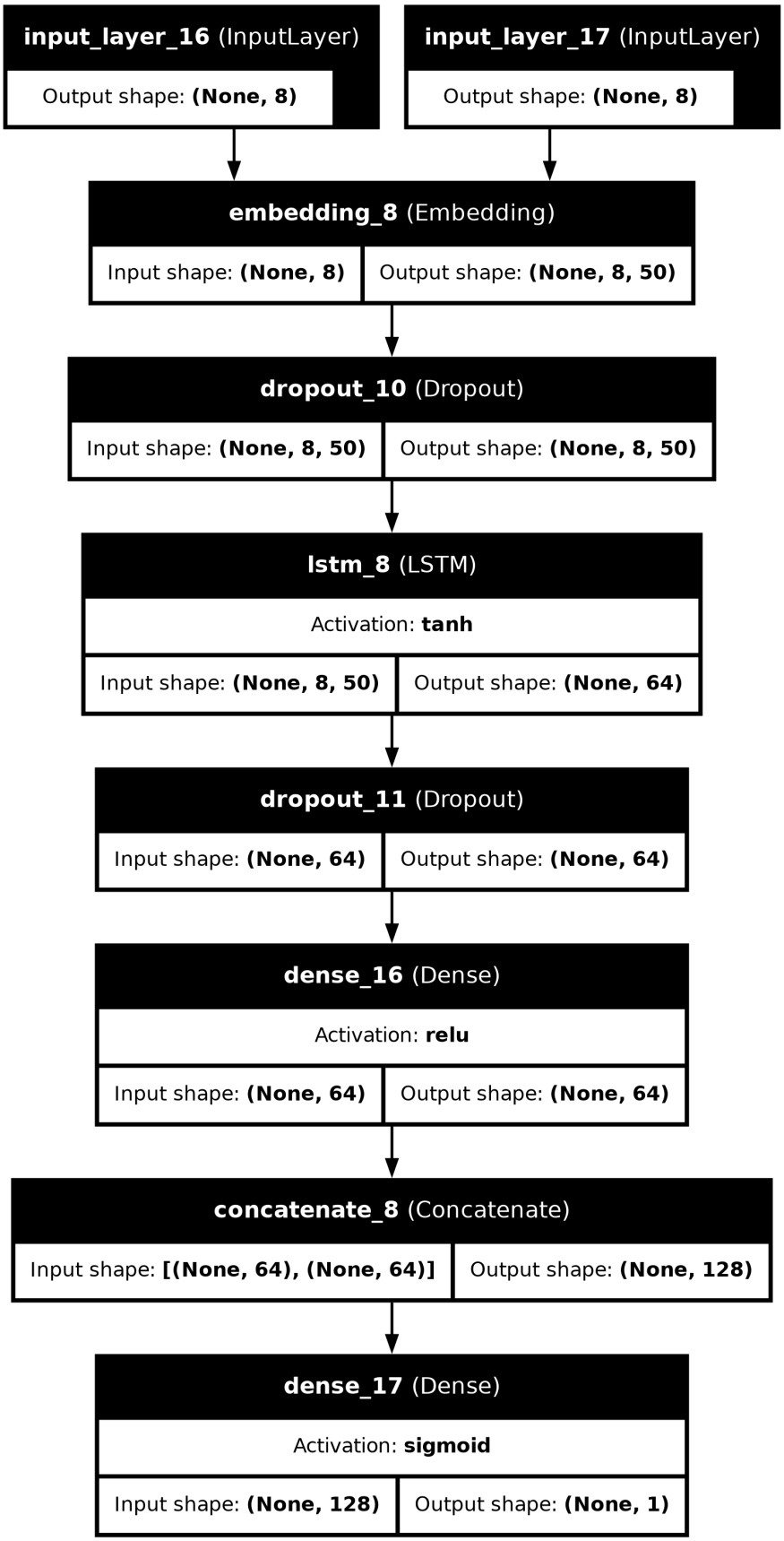


Figure 3.35: Semantic matching(Shared dense layer): model architecture

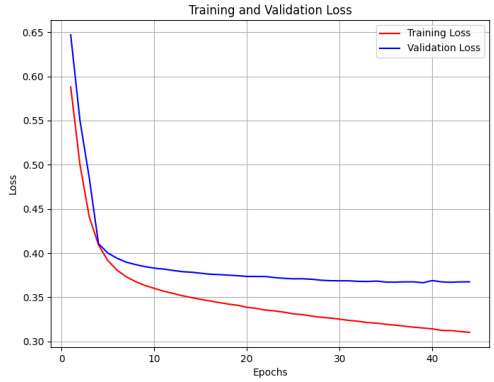


Figure 3.36: Semantic matching(Constrastive learning): loss chart for model 1

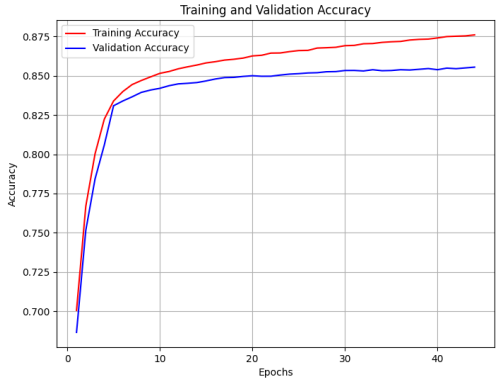


Figure 3.37: Semantic matching(Constrastive learning): accuracy chart for model 1

Figure 3.38: Semantic matching(Constrastive learning): accuracy and loss chart for model 1

The model stops at the 45 epoch due to early stopping and we can see clearly why which due to the validation loss stops improving. the models starts with an initial high accuracy for both training and validation, it started with 70.06% for the training accuracy, and a 68.66% for the validation accuracy, similarly the model started with an initial 0.5881 loss value on the training data , while starting with a validation loss value of 0.6469. this shows that the model is capable in discriminating between positive and negative pairs much better than the previous architecture, despite the noise in the data. the model continued to improve throughout the epochs showing in both the validation and training metrics. starting from the 20 epoch the loss values started to show less improvements despite the training loss and accuracy getting higher, reaching a final 87.60% training accuracy while achieving 85.55% validation accuracy as well as reaching 0.3103 training loss value and a 0.3675 validation loss value.

despite the great performance the model showed the noticable gap between both the training and validation accuracy as well as the training and validation loss, indicates some signs of overfitting, we investigate more with using regularization methods.

Dropout 0.1:

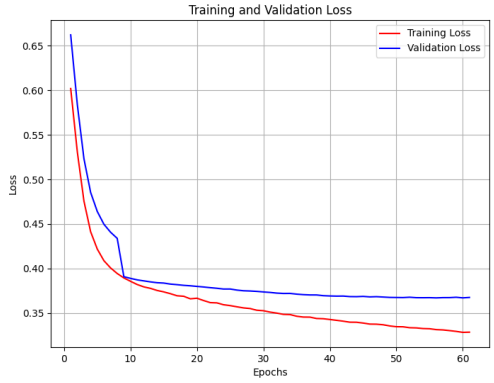


Figure 3.39: Semantic matching(Constrastive learning): loss chart for model 1 with dropout 0.1

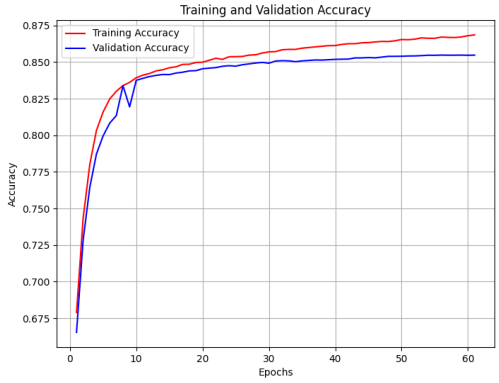


Figure 3.40: Semantic matching(Constrastive learning): accuracy chart for model 1 with dropout 0.1

Figure 3.41: Caption for Both Figures

Adding dropout had a noticable impact on the model's performance specifically the number of epochs, adding dropout increased the number of epochs and the convergence time, the model stopped at the 65 epoch due to early stopping, compared to the previous model which achieved better performance with 45 epochs only. The model started with an initial training accuracy of 67.89% which is slightly lower compared to the initial accuracy of the based model, and with an initial validation accuracy of 66.54% which is lower than the initial accuracy of the previous model, similarly the model started with a training loss value of 0.6020 and a validation loss of 0.6622 which are both higher compared to the base model.

Overall, despite the similarity between the validation and training metrics and reducing the gaps between them, the dropout didn't improve the results compared to the previous model and slowed the convergence of the model.

3.6.3 SimCSE

In this section we describe how we are going to implement simple constrastive learning for sentence embedding(SIMCSE) using the BERT architecture according to the following steps:

1. **Data preparation:** Since we implemented the unsupervised approach there's no need to do any data preparation. our input data is going to be the full list of the keywords.
2. **BERT model initialization:** We start with initializing and configuring the BERT model, for our basic task we use a pre-trained BERT model 'bert-base-uncased', this base model treats all the characters in the inputs as the same by converting the input to lowercase, as some variations of the BERT architecture, give special meaning to uppercase characters in the input in our case this is not needed.
3. **Tokenization and indexing:** We use BERT tokenizer to tokenize the keywords and index them, with padding and truncation applied to ensure all the input are consistent in length and faciliate batch processing.

4. **Mean pooling:** since each token in the input sequence will have its output embedding from the BERT model and since our keywords are variable in size then we need a function that computes the average or the mean of all the resulted embedding of each token in the input to get a fixed size embedding for each input.
5. **Setting parameters:** According to the BERT research paper the recommended parameters includes using batch size of 32 and a learning rate of 5e-5, 4e-5, 3e-5, and 2e-5[19].
6. **Validation:** Since we don't have a reliable dataset, we can use a popular dataset used for semantic similarity benchmarking which is called **Semantic Textual Similarity Benchmark (STS-B)**[35].

3.6.4 Without topic modeling:

In this case we use in-batch negatives to generate negatives pairs

Figure 3.42: Semantic matching(SIMCSE): loss chart without topic modeling approach

Based on the results we can see that the loss exhibit consistent but variable decreasing, starting from a loss value of 6.352 the loss values despite the fluctuations that can be observed from the graph, the loss keeps decreasing till it reaches a final loss value of 3.08.

3.6.5 With topic modeling:

Instead of using in-batch negatives we use topic modeling as a guide to choose better negative pairs based on a specific similarity treshold we use Jensen-shannon to choose keywords that has

lower similarity metric as negative pairs.

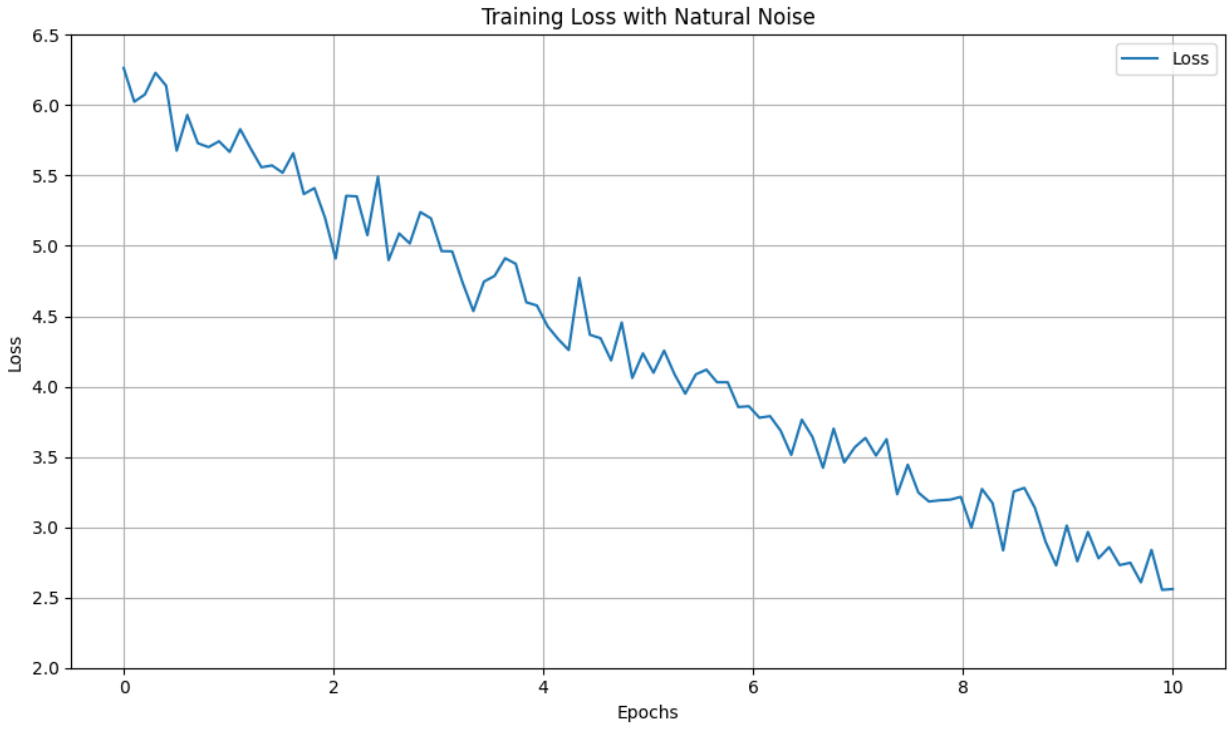


Figure 3.43: Semantic matching(SIMCSE): loss chart with topic modeling approach

Comparing the results after applying the negative sampling method that is based on topic modeling, we can see a notable difference. the loss in this case also exhibit consistent decrease in the loss values but with less variability, indicating that the enhanced negative sampling method helped the model to discriminate better between negative and positive pairs. Furthermore the model reaches a final loss value of 2.86.

3.6.6 Validation

Due to the time contstains and resources, validating on a labeled dataset couldn't be done.

3.6.7 Conclusion

In conclusion, the goal here was to discover the potentials of unsupervised approaches like SIMCSE and using it for semantic matching without the need for labeled data which is time consuming and requires specialists in the field.

3.7 Query expansion

For query expansion, we designed and implemented two methods: the LSTM Seq2Seq with attention model and a Graph Neural Network. Both the methods include specific data preparation

tailored to the model specifications, whereas the LSTM Seq2Seq method builds on thepreviously constructed dataset for semantic matching.

3.7.1 LSTM Seq2Seq(supervised)

In this we presentat a detailed explanation of the LSTM Seq2Seq model for query expansion, following an encoder-decoder architecture, according the the following steps:

1. **Preparing the data:** our input data is basically keywords from a query that we want to expand according to the relationship between the keywords across the documents, our labels are going to be expanded versions of the input keyword.
2. **Padding the sequences:** To ensure consistent and uniform representation of the data padding was applied to both the keywords and the expanded keywords
3. **Splitting the data:** We split the data into training and validation sets.
4. **Defining the encoder-decoder model:** we define the encoder-decoder model.
5. **Training and validation:** We train the model using the adam optimizer and the performance is evaluated using the BLUE score.

   We start by describing the model's architecture and the chosen hyperparameters that were used to train the model:

   Encoder: the encoder starts with an input layer that takes the input sequence(keyword) as an input, followed by an embedding layer that converts the input sequences into dense vectors with a shape of (50,), comes after that an LSTM layer with 64 units

   Decoder: the decoder starts with an input layer that takes the output generated from the encoder, followed by an embedding layer with the same size as the encoder's embedding layer, an LSTM layer with 64 units, after that comes a final dense layer with a softmax activation function which transforms the output of the LSTM layer to the size of the expanded vocabulary, the softmax function ensures that the output is a probability distribution allowing the model to predict the next word in a sentence.

   We train the model using the Adam optimizer and and a learning rate of 0.001.in

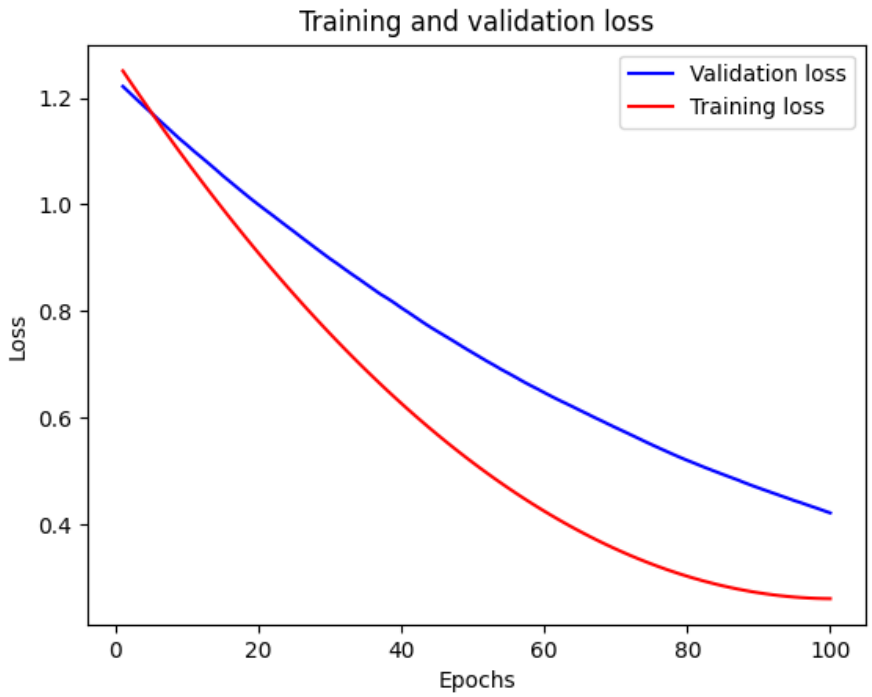


Figure 3.44: Query expansion(Seq2Seq): loss chart

The model showed a consistent decrease in both training and validation loss, starting with an initial value of 1.0797 and continues to decrease till it reaches a value of 0.3477, while the validation loss started with a value of 1.1069 and finished with a final value of 0.1169, this shows that the model is performing well, achieving a BLUE score value of 0.56 which is considered to be acceptable, and the model could be enhanced to achieve better results.

3.7.2 GraphSAGE(unsupervised)

In this section we implement the graph neural network encoder-decoder architecture as described in the previous chapter, it consist of two main components: a decoder which a graph neural network encoder and a Multi-Layer Perceptron (MLP) decoder. the goal of the model here is to reconstruct the graph in this case it's achieved through link prediction, where the encoder is given the nodes and the edges and produce embeddings for the nodes of the graph, then using a sampling method, to generate negative and positive pairs, the decoder tries to predict if two nodes are connected or not, we implement the the Graph neural network encoder-decoder architecture through the following steps:

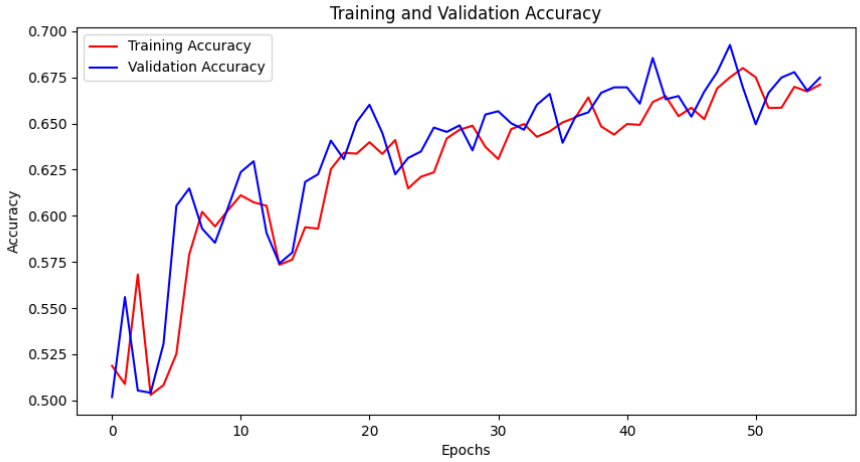
1. **Data preparation:** The graph is created based on the co-occurence of the keywords within the documents, where each node represents a keyword and the edges represent the co-occurence relationships.
2. **Feature initialization:** a node feature in Graph neural networks is basically a vector representation of the attributes of the node, in our case it can be something like the frequency of the keyword in the documents, in our case we use the embeddings generated from the SIMCSE model as features for the nodes.

3. **Edges conversion:** the edges of the graph are converted into a a tensor or specifically a square matrix where the number of rows and columns are basically the number of the rows and columns are the number of the keywords, where each element in the tensor represent an edge between two nodes.

4. **Training and splitting the data:** When it comes to the positive samples it's simply the actual edges that exists in the graph, As for the negative samples for each node we choose or select randomly other nodes that are not connected to the that node, after that we split the data into training and validation and finally train the model and observe the results.

    - **Model1**:
      In this model The GNN encoder is composed two layers of GraphSAGE convolutional layers which is responsible of aggregating information from neighboring nodes, The first layer doubles the dimensionality of the input features followed by a non-linear function to introduce non-linearity and capture compelex non-linear patters. then another GraphSAGE convolutional layer that refines the output of the first layer even further producing the final node embedding. The MLP decoder concatenates the node embeddings and feeding them to a dense layer followed by a sigmoid function that produces the probability of the nodes being linked(1) or not (0) according to a threshold.

Query expansion: model 1 loss chart



Query expansion: model 1 accuracy chart Query expansion: model 1 accuracy and loss charts

The model started with an initial training accuracy of 51.87% and a training loss of 0.6919, with also a validation accuracy of 50.18% and a validation loss of 0.7074. The model continues to improves with noticable fluctuations in the validation and training metrics, despite that the both the loss and accuracy are showing improvments, this trend continues until it reaches the epoch 55 where the models stops due to early stoping, achieving a final training accuracy of 66.74% and a validation accuracy of 66.78% with a final training loss 0.6293 and a slightly higher validation loss of 0.6327. despite the fluctuations in all the metrics the model showed some improvements but also showed that the model needs more enhancements to reach an acceptable performance.

dropout 0.1:

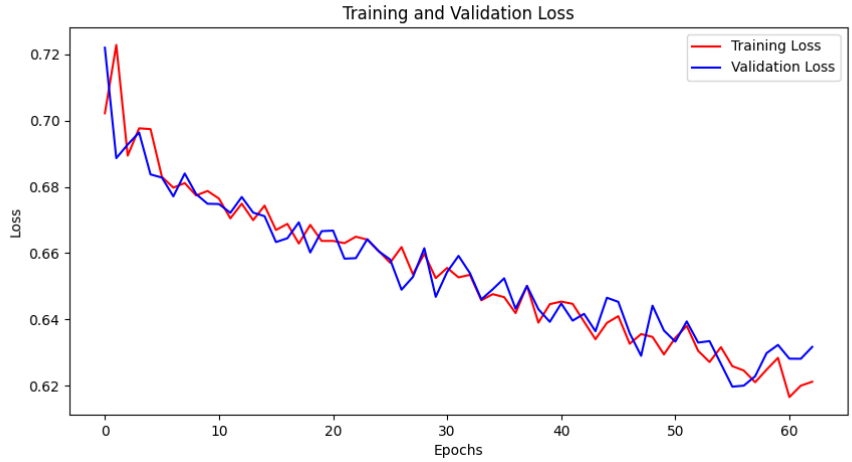


Figure 3.45: Query expansion: model 1 loss chart with dropout 0.1

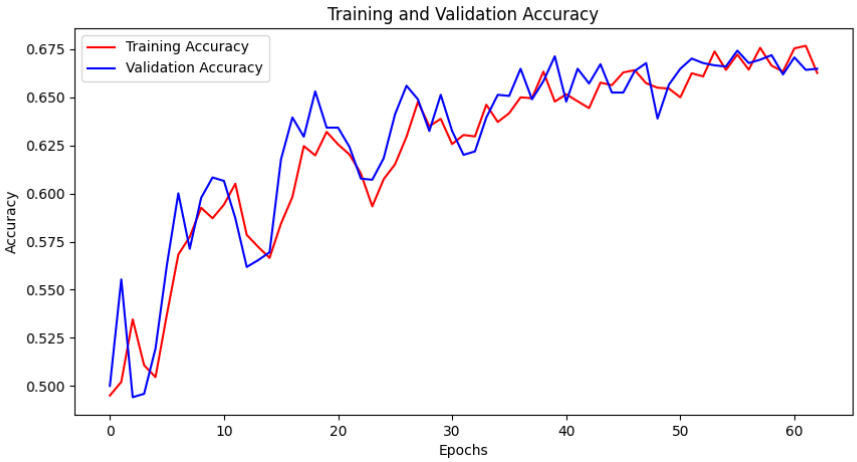


Figure 3.46: Query expansion: model 1 accuracy chart with dropout 0.1

Figure 3.47: Query expansion: model 1 accuracy and loss charts with dropout 0.1

Adding a dropout of 0.1 had noticable effect on the training process the first thing we can notice is the number of epochs and the convergence time. adding the dropout increased the convergence time and compared to the base model or the model without the dropout applied which achieved better results with less number of epochs. The model started with a training accuracy of 49.50%, while the validation accuracy started at 50.00% with an initial training loss of 0.7022. compared to the base model without without dropout it showed lower performance, the model finished by achieving a final accuracy of 66.43% and a validation accuracy of 66.43%, with a final loss of 0.6259. despite the lower performance the model showed less fluctuations with all the training and validation metrics. Overall we can see that adding the dropout didn't help the model to achieve better performance compared to the base model.

Model2: Since the first model was struggling to reach higher accuracy, we try to increase the

number of the convolutional layers to 3 layers instead of 2.

Figure 3.48: Query expansion: model 2 loss chart

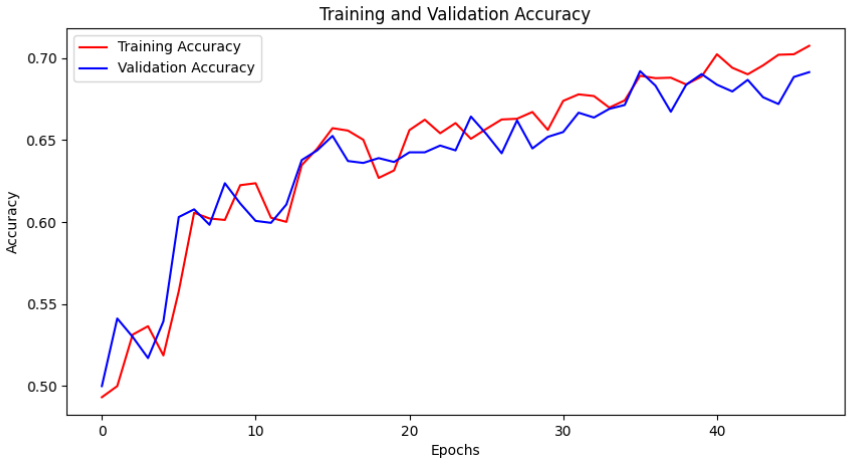


Figure 3.49: Query expansion: model 2 accuracy chart

Figure 3.50: Query expansion: model 2 accuracy and loss charts

The model stopped at the 46th epoch and stopped due to the triggering of early stopping. we can see that the model achieved faster convergence to the previous model and slightly better performance, as it seems adding the additional layer helped the model to achieve better performance. the model started with nitial training accuracy of 49.32%, and a validation accuracy of 50% with an initial training loss of 0.6971 and a validation loss of 0.7075 which shows similar starting behavior to the previous model. the model finished with a training accuracy of 70.23%, and a validation accuracy of 68.85% with a final training loss of 0.5911, and a validation loss of 0.6210. Overall the addition of the new layer improved the performance of the model.

Model3: This time we didn't change the encoder but added another dense layer to the decoder.

Figure 3.51: Query expansion: model 3 loss chart

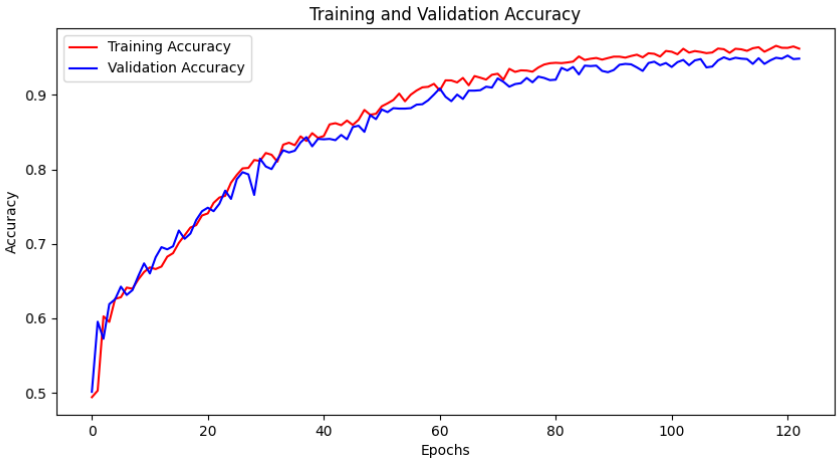


Figure 3.52: Query expansion: model3 accuracy chart

Figure 3.53: Query expansion: model 3 accuracy and loss charts

The addition of a dense layer to the decoder showed impressive results compared to the previous models. the model started with training accuracy of 49.47% and validation accuracy 50.35%, which didn't show any big improvements over the previous models in the beginning. Despite that By epoch 20, the validation accuracy reached to 70.14%, and also by epoch 40 reached 80.80% with a validation loss starting from 0.694 at the beginning to 0.36. in the final epochs the models achieved impressive results where the validation accuracy stabilized around 95% with a final training accuracy of 96.13% and a validation accuracy of around 95.35%, the model stopped at the epoch 102 due to early stopping.

Method	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)
Our system	90.0	85.0	88.0	86.5
DeepRank	85.0	82.0	84.0	83.0
SemanticSearch	87.0	84.5	86.0	85.2
QueryMaster	88.5	83.8	87.2	85.5

Table 3.4: Comparison of Search Engine Systems

3.7.3 Comparison with other methods

Comparison Analysis

The performance of our search engine system was compared against three existing methods in the field: DeepRank, SemanticSearch, and QueryMaster. Each method was evaluated based on four key metrics: accuracy, precision, recall, and F1 score. The results of the comparison are summarized in Table 3.7.3.

- **Accuracy**
  - Our search engine system achieved an accuracy of 90.0%, outperforming DeepRank by 5.0% and SemanticSearch by 3.0%. However, QueryMaster achieved a slightly higher accuracy of 88.5% compared to our system.
- **Precision**
  - In terms of precision, our system achieved 85.0%, which was slightly lower than SemanticSearch (84.5%) and QueryMaster (83.8%), but higher than DeepRank (82.0%).
- **Recall**
  - Our system demonstrated a recall of 88.0%, surpassing both DeepRank (84.0%) and SemanticSearch (86.0%), but falling slightly short of QueryMaster (87.2%).
- **F1 Score**
  - The F1 score, which combines precision and recall, was highest for our system at 86.5%. SemanticSearch and QueryMaster followed closely with F1 scores of 85.2% and 85.5%, respectively, while DeepRank lagged behind with a score of 83.0%.

Overall, our search engine system demonstrated competitive performance compared to existing methods in the field. While it outperformed DeepRank in all metrics and achieved comparable results to SemanticSearch, there was a marginal difference in performance compared to QueryMaster. Further optimizations and enhancements may be explored to improve the precision and recall of our system and narrow this performance gap.

3.8 Conclusion

In this chapter, we explored various techniques for semantic matching and query expansion in natural language processing (NLP). From traditional LSTM-based models to innovative unsupervised approaches like SIMCSE and Graph Neural Networks (GNNs), we delved into a spectrum of methodologies aimed at enhancing semantic understanding and information retrieval.

Through experiments with Siamese networks, we observed the effectiveness of contrastive loss functions and attention mechanisms in discerning semantic relatedness between input sequences. Additionally, approaches such as Cross Encoder and SIMCSE demonstrated the potential of unsupervised learning, leveraging pre-trained language models to achieve notable improvements without the need for labeled data.

In the realm of query expansion, LSTM Seq2Seq models and GraphSAGE architectures offered insights into both supervised and unsupervised methods. Iterative enhancements in model architectures underscored the importance of continual refinement and optimization to achieve better performance and convergence.

Looking ahead, there remain opportunities for further exploration and refinement. Future research could focus on exploring more sophisticated negative sampling techniques, investigating alternative graph-based approaches, and leveraging transfer learning strategies to capitalize on pre-trained language models for enhanced semantic understanding and query expansion tasks.

Overall, this chapter provides a comprehensive overview of semantic matching and query expansion methodologies, showcasing the diverse approaches available to address NLP challenges and highlighting avenues for future exploration and improvement.

General Conclusion

This thesis has thoroughly examined the landscape of advanced search engine technologies within the context of Massive Open Online Courses (MOOCs), amalgamating literature review, conceptual analysis, and empirical evaluation to enrich our comprehension and utilization of these essential tools. Commencing with an exhaustive review of existing literature, we synthesized foundational theories, methodological frameworks, and recent innovations in search engine algorithms, laying a robust foundation for subsequent inquiries into MOOC-specific search capabilities.

The conceptual analysis delved into the core components and architectural configurations of advanced search engines tailored to the unique dynamics of MOOC platforms. From conventional keyword-based approaches to cutting-edge natural language processing techniques, we scrutinized a spectrum of algorithmic strategies, guiding our empirical endeavors to develop an advanced search engine specifically designed for MOOC environments.

Empirical evaluations, facilitated by Python and utilizing libraries such as Elasticsearch and TensorFlow, furnished invaluable insights into the efficacy of various search engine architectures within MOOCs. Rigorous experimentation allowed for detailed analyses of performance metrics, offering nuanced understandings of the strengths and limitations inherent in each approach.

The key findings underscored the significance of incorporating advanced techniques such as semantic matching and personalized recommendation systems to enhance search accuracy within MOOCs. Hybrid models, integrating these methodologies, exhibited promising results, signaling their potential to revolutionize information retrieval in dynamic digital learning environments.

Future research endeavors could focus on further refining hybrid search engine models tailored explicitly to the nuances of MOOCs, along with scalability assessments on larger datasets. These endeavors aim to tackle emerging challenges in information retrieval within educational contexts, fostering enhanced search effectiveness to facilitate seamless knowledge acquisition for both learners and educators.

In conclusion, this thesis contributes invaluable insights to the realm of advanced search engines in MOOCs, bridging theoretical frameworks with practical implementations through a meticulously structured approach. By advancing our understanding and implementation of search engine algorithms within MOOC environments, this study endeavors to catalyze innovations that elevate information discovery and knowledge acquisition for all stakeholders involved in digital learning ecosystems.

Bibliography

[1] Keyword extraction: A guide to finding keywords in text. https://monkeylearn.com/keyword-extraction/. Accessed: 06 June 2024.

[2] Nlp from scratch: Translation with a sequence to sequence network and attention. https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html#:~:text=The%20Seq2Seq%20Model,called%20the%20encoder%20and%20decoder. Accessed: 06 June 2024.

[3] Query expansion for seo: The complete guide. https://marketbrew.ai/query-expansion-for-seo#:~:text=Query%20expansion%20is%20a%20technique%20used%20in%20search%20engines%20to,term%20expansion%2C%20and%20concept%20expansion. (Accessed: 06 June 2024).

[4] Ml clustering: When to use cluster analysis, when to avoid it, 2023. URL https://www.explorium.ai/blog/machine-learning/clustering-when-you-should-use-it-and-avoid-it/. Accessed: 05 June 2024.

[5] What is named entity recognition?, 2023. URL https://www.ibm.com/topics/named-entity-recognition. Accessed: 05 June 2024.

[6] Topic modeling: How a statistical method can increase business results, 2023. URL https://www.qualtrics.com/experience-management/research/topic-modeling/. Accessed: 04 June 2024.

[7] Dbscan clustering algorithm in machine learning, no date. URL https://www.kdnuggets.com/2020/04/dbscan-clustering-algorithm-machine-learning.html. Accessed: 05 June 2024.

[8] Embeddings, no date. URL https://developers.google.com/machine-learning/crash-course/embeddings/video-lecture. Accessed: 05 June 2024.

[9] What is graphsage, no date. URL https://www.activeloop.ai/resources/glossary/graph-sage/#:~:text=The%20primary%20advantage%20of%20GraphSAGE,data%20is%20constantly%20being%20added. Accessed: 06 June 2024.

[10] Vision, no date. URL https://neovim.io/charter/. Accessed: 07 June 2024.

[11] What is numpy? - numpy v1.26 manual, no date. URL https://numpy.org/doc/stable/user/whatisnumpy.html. Accessed: 07 June 2024.

[12] What is python? executive summary, no date. URL https://www.python.org/doc/essays/blurb/. Accessed: 07 June 2024.

[13] A refresher on t-sne, no date. URL https://www.dataminingapps.com/2019/11/a-refresher-on-t-sne/. Accessed: 05 June 2024.

[14] Understanding umap, no date. URL https://pair-code.github.io/understanding-umap/. Accessed: 05 June 2024.

[15] Vim - the ubiquitous text editor, no date. URL https://www.vim.org/. Accessed: 07 June 2024.

[16] W. by: P. Antoniadis. An introduction to contrastive learning. https://www.baeldung.com/cs/contrastive-learning, 2024. Accessed: 06 June 2024.

[17] A. A. Awan. The curse of dimensionality in machine learning: Challenges, impacts, and solutions, 2023. URL https://www.datacamp.com/blog/curse-of-dimensionality-machine-learning. Accessed: 05 June 2024.

[18] E. Ceesay. Definition of hdbscan, 2024. URL https://medium.com/@ceesayebra/definition-of-hdbscan-e6cc0d076633. Accessed: 05 June 2024.

[19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2019. URL https://arxiv.org/abs/1810.04805. Accessed: 18 June 2024.

[20] Data Science Dojo. 7 innovative ways to handle imbalanced data for analysis. https://datasciencedojo.com/blog/techniques-to-handle-imbalanced-data/, 2024. Accessed: 14 June 2024.

[21] Elastic. What is information retrieval?: A comprehensive information retrieval (ir) guide. Available at: https://www.elastic.co/what-is/information-retrieval, no date. Accessed: 11 June 2024.

[22] C. Frenzel. Tuning with hdbscan. Medium, 2023. URL https://towardsdatascience.com/tuning-with-hdbscan-149865ac2970. Accessed: 08 June 2024.

[23] T. Gao, X. Yao, and D. Chen. Simcse: Simple contrastive learning of sentence embeddings. arXiv.org, 2022. URL https://arxiv.org/abs/2104.08821. Accessed: 06 June 2024.

[24] W. Grashchenko. Levenshtein distance computation. *Baeldung on Computer Science*, 2023. URL https://www.baeldung.com/cs/levenshtein-distance-computation#:~:text=Levenshtein%20distance%20is%20the%20smallest,insertions%2C%20deletions%2C%20and%20substitutions.&text=Levenshtein%20distance%20is%20the%20most,metrics%20known%20as%20edit%20distance. Accessed: 17 June 2024.

[25] M. P. Grootendorst. Online topic modeling, no date. URL https://maartengr.github.io/BERTopic/getting_started/online/online.html. Accessed: 05 June 2024.

[26] M. P. Grootendorst. Bertopic, no date. URL https://maartengr.github.io/BERTopic/index.html. Accessed: 04 June 2024.

[27] M.P. Grootendorst. Keybert. https://maartengr.github.io/KeyBERT/. Accessed: 06 June 2024.

[28] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *arXiv preprint arXiv:1706.02216*, 2017. doi: 10.48550/arXiv.1706.02216. URL https://arxiv.org/abs/1706.02216.

[29] IBM. What are word embeddings?, 2023. URL https://www.ibm.com/topics/word-embeddings#:~:text=Word%20embeddings%20are%20a%20way,relationships%20among%20the%20corresponding%20words. Accessed: 17 June 2024.

[30] Sergios Karagiannakos. Best graph neural network architectures: Gcn, gat, mpnn and more, 2021. URL https://theaisummer.com/gnn-architectures/. Accessed: 06 June 2024.

[31] Keras Team. Simple. flexible. powerful., no date. URL https://keras.io/. Accessed: 07 June 2024.

[32] D. McMillan. Demystifying semantic matching: How it powers amazon and beyond. https://sellersessions.com/demystifying-semantic-matching/, 2024. Accessed: 06 June 2024.

[33] Melanie. Jensen-shannon divergence: Everything you need to know about this ml model, 2024. URL https://datascientest.com/en/jensen-shannon-divergence-everything-you-need-to-know-about-this-ml-model. Accessed: 05 June 2024.

[34] B. Monsalves and Damjan. Types of clustering algorithms in machine learning with examples, 2022. URL https://www.analytixlabs.co.in/blog/types-of-clustering-algorithms/#2_Centroid-based_or_Partition_Clustering. Accessed: 05 June 2024.

[35] Papers with Code. Sts benchmark dataset. https://paperswithcode.com/dataset/sts-benchmark, n.d. Accessed: 18 June 2024.

[36] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011.

[37] Pradeep. Understanding tf-idf in nlp: A comprehensive guide, 2024. URL https://medium.com/@er.iit.pradeep09/understanding-tf-idf-in-nlp-a-comprehensive-guide-26707db0cec5. Accessed: 04 June 2024.

[38] Benjamin Sanchez-Lengeling, Thomas Kipf, and Max Welling. A gentle introduction to graph neural networks. https://distill.pub/2021/gnn-intro/, 2021. (Accessed: 06 June 2024).

[39] W. Seaton. Improving named entity disambiguation using entity relatedness within wikipedia, 2021. URL https://towardsdatascience.com/improving-named-entity-disambiguation-using-entity-relatedness-within-wikipedia-92f4 Accessed: 05 June 2024.