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

University of 8 May 1945-Guelma-

Faculty of Mathematics, Computer Science and Science of Matter
Department of Computer Science

Master Thesis

Specialty : Computer Science

Option : Information Systems

Theme :

A method based on word embedding and semantic similarity for detecting aberrant documents

Presented by : Adjabi Akram

Jury Members :

N	Full Name	Quality
1	Dr. Chouhra Chemes Eddine	Chairman
2	Dr. Farek Lazhar	Supervisor
3	Dr. Tadjer Houda	Examiner

June 2024

Acknowledgements

It is often said that the journey is as important as the destination. The five years I spent at university allowed me to fully understand the meaning of this simple phrase. This journey, in fact, has not been achieved without challenges and without raising numerous questions for which the answers require long hours of work.

At the end of this work, I would like to thank "Allah" the Almighty for giving me faith and allowing me to get to this point.

I would like to thank from the bottom of my heart my parents "Salah" and "Radia" for their moral support throughout my years of study and all the sacrifices they made to allow me to follow my studies in the best conditions. possible and never stopped encouraging me.

I would like to express my deep gratitude to **Dr.Farek Lazhar**, for supervising and directing my research. I thank him for and supporting me throughout my thesis. His valuable advice, his rigor and his comments made it possible to greatly improve the quality of my work and this dissertation. Honestly, thanks to him, I was able to learn a lot of things, some of which were very useful for my academic work of course, but also important things for my personal development. Finally, I have not forgotten his precious help in proofreading and correcting my thesis.

I also thank my sisters and brothers who have always encouraged and supported me morally. And to all my friends who supported me directly or indirectly, a big thank you.

Finally, I thank all the professors of the computer science department of the University of May 8, 1945 of Guelma

Dedication

To my mother, whose unwavering dedication and unconditional support have been the cornerstone of my success. Through her boundless love, sacrifices, and invaluable guidance, she has shaped my academic journey and encouraged me to pursue my dreams. Through this work, I wish to convey my eternal gratitude to her for her constant presence in my life and for being my source of strength and inspiration.

To my father, whose perseverance and sacrifice have been an endless source of inspiration. His years of hard work and dedication have paved the way for my success. May God watch over him, and may this work be a testament to his unwavering support and noble values that have guided me throughout my journey.

Abstract

Detecting outlier documents is a critical task in various domains, including fraud detection, information retrieval, and anomaly detection. This project leverages Word2Vec Framework and the Word Mover's Distance (WMD) to identify outlier documents in a corpus. Word2Vec is utilized to generate dense vector representations of words, capturing semantic similarities and contextual relationships. The WMD, which measures the dissimilarity between two text documents by computing the minimal cost to transform one document into another, is applied to these vector representations to assess document similarity. By analyzing the distribution of WMD scores across the document corpus, we can identify documents that deviate significantly from the norm, thus classifying them as outliers. This approach is advantageous due to its ability to handle the semantic richness of text and provide a nuanced measure of document similarity. The effectiveness of the proposed method is validated through experiments on benchmark datasets, demonstrating its potential in accurately identifying outlier documents.

Keywords: Outlier Detection, Word2Vec, Word Mover's Distance (WMD), Document Similarity, Anomaly, Detection, Semantic Analysis, Text Mining, Vector Representations, Information Retrieval, Natural Language Processing (NLP).

Résumé

La détection de documents atypiques est une tâche critique dans divers domaines, y compris la détection de fraude, la recherche d'information et la détection d'anomalies. Ce projet utilise le framework Word2Vec et Word Mover's Distance (WMD) pour identifier les documents aberrants dans un corpus. Word2Vec génère des représentations vectorielles denses des mots, capturant les similarités sémantiques et les relations contextuelles. La WMD, qui mesure la dissimilarité entre deux documents en calculant le coût minimal pour transformer un document en un autre, est appliquée à ces représentations vectorielles pour évaluer la similarité des documents. En analysant la distribution des scores de la WMD à travers le corpus, nous pouvons identifier les documents qui s'écartent significativement de la norme et les classer ainsi comme atypiques. Cette approche est avantageuse en raison de sa capacité à gérer la richesse sémantique des textes et à fournir une mesure nuancée de la similarité des documents. L'efficacité de la méthode proposée est validée par des expériences sur des ensembles de données de référence, démontrant son potentiel à identifier avec précision les documents hors norme.

Mots-clés : Détection d'anomalies, Word2Vec, Word Mover's Distance (WMD), Similarité des documents, Analyse sémantique, Exploration de texte, Représentations vectorielles, Recherche d'information, Traitement automatique du langage naturel (TALN).

Contents

List of Figures

List of Tables

List of Abbreviations

AI	**A**rtificial **I**ntelligence
CBOW	**C**ontinuous **B**ag **o**f **W**ords
DNN	**D**eep **N**eural **N**etworks
FN	**F**alse **N**egative
FP	**F**alse **P**ositive
IDE	**I**ntegrated **D**evelopment **E**nvironment
KDD	**K**nowledge **D**iscovery in **D**atabases
NLP	**N**atural **L**anguage **P**rocessing
nltk	**N**atural **L**anguage **T**oolkit
TP	**T**rue **P**ositive
TN	**T**rue **N**egative
VS Code	**V**isual **S**tudio **C**ode
WMD	**W**ord **M**over's **D**istance
XAI	**E**xplainable **A**rtificial **I**ntelligence

General Introduction

In the current era of data-driven decision making, the importance of detecting anomalies and outliers in datasets has become paramount across various domains. From fraud detection in finance to identifying cybersecurity threats, the ability to accurately pinpoint outliers can significantly enhance operational efficiency and security. This project explores advanced methodologies for outlier detection, leveraging state-of-the-art techniques such as Word2Vec and the Word Mover's Distance (WMD). These techniques are employed to generate dense vector representations of words, which capture semantic similarities and contextual relationships, enabling a nuanced assessment of document similarity. By analyzing the distribution of WMD scores, this project aims to identify documents that deviate significantly from the norm, classifying them as outliers. This approach not only handles the semantic richness of text but also provides a robust measure of document similarity, demonstrating its potential for effective anomaly detection.

This master thesis consists of: Abstract, General Introduction, General Conclusion and Four Chapters:

— **Chapter 1. Outlier Detection** - We present the concept of outlier detection, its significance, and various traditional and modern methods.

— **Chapter 2. Word Embedding Techniques** - Delves into word embedding techniques, comparing traditional methods with advanced models like Word2Vec and FastText.

— **Chapter 3. Conceptual Framework** - Presents the conceptual framework, including theoretical foundations and the application of Word Mover's Distance (WMD) for text similarity.

— **Chapter 4. Implementation** - Details the implementation process, from development environment setup to data preprocessing, model training, and evaluation metrics.

Chapter 1

Outlier Detection

1.1 Introduction

Anomalies within data are distinct patterns that deviate from the norm, often providing valuable insights by indicating hidden threats, unexpected opportunities, or unique occurrences. Anomaly detection, the process of identifying these deviations, is essential in various fields, including finance and healthcare.

This chapter introduces the concept of anomaly detection, discussing its significance and the challenges associated with it. We will explore different types of anomalies, ranging from isolated points to collective patterns, and address the complexities of defining normalcy, adapting to adversarial conditions, and managing shifting baselines. Through this exploration, we will highlight the potential of anomaly detection to reveal significant insights from data.

Additionally, this chapter will examine advanced techniques used in anomaly detection and their applications across different domains. By understanding these techniques and their implications, we gain a deeper appreciation of how anomaly detection influences our understanding of data and its impact on various aspects of life.

1.2 Definitions

Outliers, also known as anomalies, have been defined and conceptualized in diverse ways across various domains and disciplines. A clear understanding of these definitions is essential for the effective detection and management of outliers in data analysis .

Below, we present some commonly encountered definitions from the literature, along with examples:

1.2.1 Deviation from the Mean

Outliers are extreme data points that are beyond the expected norms for their type. This can be a whole data set that is confounding, or extremities of a certain data set. Imagining a standard bell curve, the outliers are the data on the far right and left. These outliers can indicate fraud or some other anomaly you are trying to detect, but they can also be measurement errors, experimental problems, or a novel, one-off blip. Basically, it refers to a data point or set of data points that diverges dramatically from expected samples and patterns.

There are two types of outliers, multivariate and univariate. Univariate outliers are a data point that is extreme for one variable. A multivariate outlier is a combination of unusual data points, including at least two data points.

a) Point outliers: These are single data points that are far removed from the rest of the data points.

b) Contextual outliers: These are considered to be 'noise', such as punctuation symbols and commas in text, or background noise when performing speech recognition.

b) Collective outliers: These are subsets of unexpected data that show a deviation from conventional data, which may indicate a new phenomenon [1].

1.2.2 Another definition

An outlier may be defined as a piece of data or observation that deviates drastically from the given norm or average of the data set. An outlier may be caused simply by chance, but it may also indicate measurement error or that the given data set has a heavy-tailed distribution [2].

1.2.3 Understanding Outlier Detection

Outlier detection, also known as anomaly detection, is a statistical technique used to identify observations that deviate significantly from the majority of data. An outlier is an observation that lies an abnormal distance from other values in a random sample from a population. In a sense, outliers are data points that do not adhere to the common statistical patterns and trends exhibited by the majority of data points.

Outliers can arise due to various reasons, including measurement or input error, data corruption, or they can be genuine observations that are simply rare or represent a new trend. In any case, outlier detection is crucial because outliers can lead to significant inaccuracies in data analysis and predictive modeling [3].

1.3 Types of Outliers

An important aspect of an anomaly detection technique is the nature of the desired anomaly. Different techniques are better suited for identifying different types of anomalies. Anomalies can be broadly classified into three main categories:

1.3.1 Point Anomalies

Point Anomalies: These are the most straightforward outliers, representing individual data points that deviate significantly from the rest of the data. Imagine a scatterplot where most points cluster in the center, but a few outliers appear far away. These isolated points could represent anything from a credit card transaction far exceeding a user's typical spending habits to a medical test result outside the expected range. Identifying point anomalies is often the focus of many anomaly detection algorithms [4].

1.3.2 Contextual Anomalies

If a data instance is anomalous in a specific context (but not otherwise), then it is termed as a contextual anomaly (also referred to as conditional anomaly The notion of a context is induced by the structure in the data set and has to be specified as a

part of the problem formulation. Each data instance is defined using following two sets of attributes:

a) Contextual attributes: The contextual attributes are used to determine the context (or neighborhood) for that instance. For example, in spatial data sets, the longitude and latitude of a location are the contextual attributes. In timeseries data, time is a contextual attribute which determines the position of an instance on the entire sequence.

b) Behavioral attributes: The behavioral attributes define the non-contextual characteristics of an instance. For example, in a spatial data set describing the average rainfall of the entire world, the amount of rainfall at any location is a behavioral attribute [5].

1.3.3 Collective Anomalies

Anomalies can also occur in groups, forming a pattern known as a collective anomaly In these cases, the individual data points within the group might appear unremarkable on their own. However, their combined presence paints a suspicious picture. Imagine a data set tracking credit card transactions (behavioral attribute). A single purchase for a large amount might not necessarily be anomalous. However, a surge in new accounts (contextual attribute) originating from the same region, each making a similar, but not individually suspicious, purchase, could suggest coordinated fraudulent activity This is a classic example of a collective anomaly.

Here's another scenario: In a network intrusion detection system, individual attempts to access a specific file might not raise any red flags on their own. However, a cluster of such attempts originating from different IP addresses (contextual attribute) within a short time frame (contextual attribute) could indicate a coordinated port scanning attack . Detecting collective anomalies requires algorithms that can recognize patterns within a group of data points, even if the individual points themselves don't appear particularly unusual. These algorithms need to identify relationships and interactions between data points to uncover the collective anomaly [6].

1.4 Challenges in Outlier Detection

The pursuit of anomalies is not without its challenges. The nature of anomalies, the vastness of data, and the ever-evolving landscape of threats pose unique challenges that demand innovative solutions.

a) Data Volume and Complexity: The sheer volume and complexity of modern data sets can overwhelm traditional outlier detection methods, making it difficult to identify anomalies in a timely manner.

b) Evolving Anomalies: Anomalies can evolve over time, requiring detection methods to adapt and learn from new patterns This is particularly challenging in dynamic data streams, where anomalies may emerge quickly and disappear just as rapidly.

c) Context Awareness: Many anomalies are context-dependent, meaning they are only anomalous within a specific context Traditional methods may struggle to capture these contextual relationships, leading to false positives.

d) High-Dimensional Data: High-dimensional data, with numerous features, poses challenges for outlier detection, as it can be difficult to identify meaningful patterns and relationships [9].

1.5 Outlier Detection Techniques

The pursuit of anomalies, the unusual and unexpected patterns hidden within the vast expanse of data, has given rise to a diverse array of detection methods. These techniques, each tailored to specific types of anomalies and data characteristics, empower us to uncover hidden threats, gain insights into complex systems, and even make serendipitous discoveries. In this section, we'll delve into the world of outlier detection methods, exploring traditional approaches, modern machine learning techniques, and the unique challenges they address [7].

The advent of machine learning has revolutionized outlier detection, introducing sophisticated algorithms that can adapt to complex data patterns and evolving anomalies. These techniques excel in handling high-dimensional data, noisy environments, and dynamic data streams.

1.5.1 Traditional Methods

Traditional methods in anomaly detection often rely on statistical and rule-based approaches to identify data points that deviate significantly from the expected or normal behavior. Here are some traditional methods commonly used in anomaly detection [10]:

1.5.1.1 Z-Score (Standard Score)

A Z-score as illustrated by Figure 1.1, also known as a standard score, tells you how many standard deviations a specific data point (raw score) is away from the mean (average) of the data set. It essentially expresses how far a particular point deviates from the "typical" value within the dataset. The formula to calculate a Z-score is given by Equation 1.1.

$$Z = (X - \mu)/\sigma \tag{1.1}$$

where: X Tis the individual data point you're interested in. μ is the average of all the data points in the set. σ is a measure of how spread out the data is from the mean.

Example: Imagine you have a dataset of student test scores with a mean of 70 and a standard deviation of 10. A student who scored 80 would have a Z-score of: Z-score = (80 - 70) / 10 = 1. This means the student's score is one standard deviation above the average.

In conclusion, Z-scores provide a standardized way to compare data points within a dataset, regardless of the original units of measurement. They help us understand how unusual or typical a particular value is compared to the rest of the data [11].

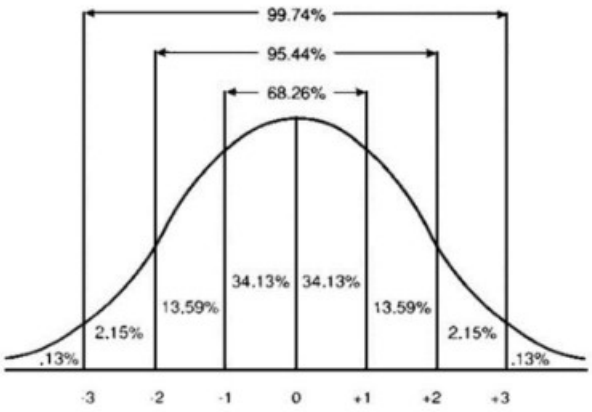

FIGURE 1.1 – Distribution of Z-Scores in a Standard Normal Distribution.

1.5.1.2 Interquartile Range (IQR)

The interquartile range (IQR) is a robust measure of variability used to identify outliers in a data set. It focuses on the middle half of the data, providing a less extreme view of spread compared to the full range. Here's a breakdown of IQR:

a) Quartiles These are partition values that divide the data set into four equal parts, with each part containing roughly 25% of the data points. There are three quartiles in total:

Q1 (Lower Quartile): Represents the value below which 25% of the data points lie.

Q2 (Median): The middle value of the data set, with 50% of the data points below it and 50% above it. (In some cases, the median might be the average of two middle values).

Q3 (Upper Quartile): Represents the value above which 75% of the data points lie.

Calculating IQR: The IQR is simply the difference between the upper and lower quartiles as show in Equation 1.2.

$$IQR = Q3 - Q1 \tag{1.2}$$

A larger IQR indicates a greater spread of data within the middle half of the distribution. This suggests there might be more outliers present in the tails of the distribution (above Q3 or below Q1). Conversely, a smaller IQR signifies a tighter clustering of data points around the middle, with fewer potential outliers.

Example:

Imagine a dataset of exam scores: {50, 65, 70, 70, 75, 80, 85, 90, 100}.

Q1 (Lower Quartile) = 65

Q3 (Upper Quartile) = 85

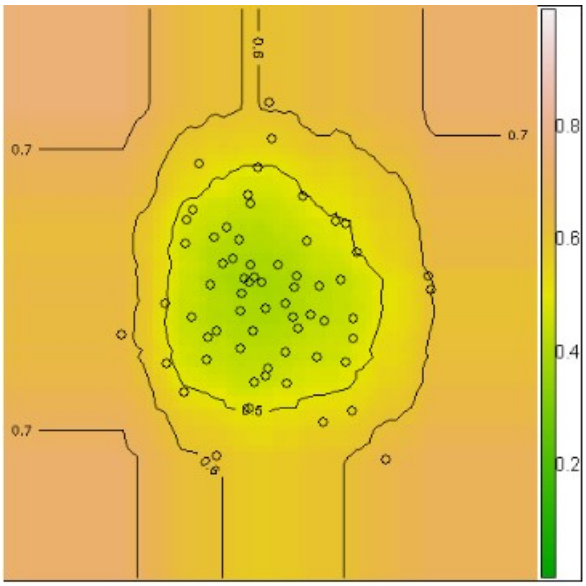

FIGURE 1.2 – Anomaly Score Contour of iForest for a Gaussian Distribution of Sixty-Four Points. Contour Lines for $s = 0.5, 0.6, 0.7$ are Illustrated. Potential Anomalies Can Be Identified as Points Where $s \geq 0.6$ [12].

IQR = Q3 - Q1 = 85 - 65 = 20

Here, the IQR is 20, indicating a moderate spread of data within the middle half of the distribution. The presence of a score as high as 100 might suggest a potential outlier in the upper tail of the data [4].

1.5.2 Machine Learning-Based Approaches

Traditional methods have laid a strong foundation for outlier detection, but modern machine learning techniques offer a powerful leap forward. These approaches leverage algorithms that can learn complex patterns and structures within data, enabling them to identify anomalies with greater accuracy and adaptability. Here, we'll delve into three prominent machine learning-based outlier detection algorithms: Isolation Forest, One-Class Support Vector Machines (OCSVM), and Local Outlier Factor (LOF) [10].

1.5.2.1 Isolation Forest

Isolation Forest (iForest) is a machine learning algorithm for anomaly detection that leverages the concept of isolation. It operates under the assumption that anomalies are rare and stand out from the normal data points. By isolating them, the algorithm can effectively identify outliers. Below is an illustrative figure:

Here's a breakdown of the Isolation Forest process:

a) Training Stage: The algorithm randomly selects a subset of data points from the training set with replacement (meaning a data point can be chosen multiple times). It then selects a random feature from the data set and creates a split rule based on that feature's value. This split rule could be a simple threshold, such as splitting the data points based on whether a specific feature value is greater than or less than a

Algorithm 1 : $iForest(X, t, \psi)$

Inputs: X - input data, t - number of trees, ψ - sub-sampling size

Output: a set of t *iTrees*

1: **Initialize** $Forest$
2: set height limit $l = ceiling(\log_2 \psi)$
3: **for** $i = 1$ to t **do**
4: $X' \leftarrow sample(X, \psi)$
5: $Forest \leftarrow Forest \cup iTree(X', 0, l)$
6: **end for**
7: **return** $Forest$

FIGURE 1.3 – iForest Algorithm.

certain value. The data points are divided according to the split rule, creating two child nodes.

This process of randomly selecting features and creating split rules continues recursively until a certain termination criterion is met, such as reaching a maximum depth for the tree or isolating a single data point in a leaf node.

This random process of creating isolation trees is repeated multiple times, resulting in a collection of isolation trees forming the Isolation Forest.

b) Testing Stage: New data points (potential anomalies) are passed through each tree in the Isolation Forest. At each split in the tree, the data point is directed to one of the child nodes based on its feature value. The number of splits (or path length) required to isolate the data point in a leaf node is recorded [13].

c) iForest Algorithm: The Isolation Forest (iForest) algorithm (Figure 1.3) is a machine learning technique used for anomaly detection. It operates under the assumption that anomalies are rare and stand out from the typical data points. By isolating these outliers, iForest effectively identifies them [13].

There are two input parameters to the iForest algorithm. They are the sub-sampling size ψ and the number of trees t. We provide a guide below to select a suitable value for each of the two parameters:

1. **Sub-sampling size ψ:** controls the training data size. We find that when ψ increases to a desired value, iForest detects reliably and there is no need to increase ψ further because it increases processing time and memory size without any gain in detection performance.
2. **Number of tree t:** controls the ensemble size. We find that path lengths usually converge well before t = 100. Unless otherwise specified, we shall use t = 100 as the default value in our experiment. At the end of the training process, a collection of trees is returned and is ready for the evaluation stage. The complexity of the training an iForest is O(tψ log ψ) [14].

1.5.2.2 Local Outlier Factor (LOF)

The Local Outlier Factor (LOF) is a technique used in anomaly detection to identify data points that are isolated or significantly different from their local neighborhoods.

Here's how it works:

1. Local Density Estimation: LOF calculates the local density of each data point by assessing the distance between that point and its neighboring points. The density is higher if a point has many neighbors close to it and lower if it has few neighbors nearby. This step helps LOF to capture the density distribution of the data in the vicinity of each point.

2. Comparison with Neighbors: Once the local density of each point is estimated, LOF compares the density of each point with that of its neighbors. If a point has a much lower density compared to its neighbors, it suggests that the point is isolated or less densely surrounded, indicating a potential outlier.

3. LOF Calculation: The LOF score for each data point is calculated based on the ratio of its local density to the average local density of its neighbors. A higher LOF score indicates that the point is more likely to be an outlier.

4. Thresholding: Based on the LOF scores, a threshold is set to identify outliers. Points with LOF scores exceeding this threshold are classified as outliers.

LOF is particularly useful in scenarios where the data is sparse or contains a significant amount of noise. Traditional distance-based methods may struggle in such cases because they rely solely on global characteristics of the data, whereas LOF considers the local context of each data point. This makes LOF robust in identifying outliers even in complex datasets where traditional methods may fail.

1.5.2.3 One-Class SVM (Support Vector Machine)

One-Class SVM (OC-SVM) is a machine learning technique used for anomaly detection. Unlike traditional SVMs that require labeled data for classification (normal vs. anomaly), OC-SVM is an unsupervised algorithm. It learns a decision function based solely on the training data, allowing it to identify new data points that deviate significantly from the normal patterns.

Advantages of OC-SVM:

— Effective for High-Dimensional Data: Can handle data sets with many features.

— Unsupervised Learning: Doesn't require labeled data for training (normal vs. anomaly).

— Flexible Boundary: Can learn complex boundaries around the normal data distribution.

Limitations of OC-SVM:

— **Tuning Parameters:** Requires careful selection of hyperparameters to achieve optimal performance.

— **Novelty Detection:** Primarily focused on identifying entirely new or unseen types of anomalies [15].

Figure 1.4 illustrates the normal distribution for One-Class SVM.

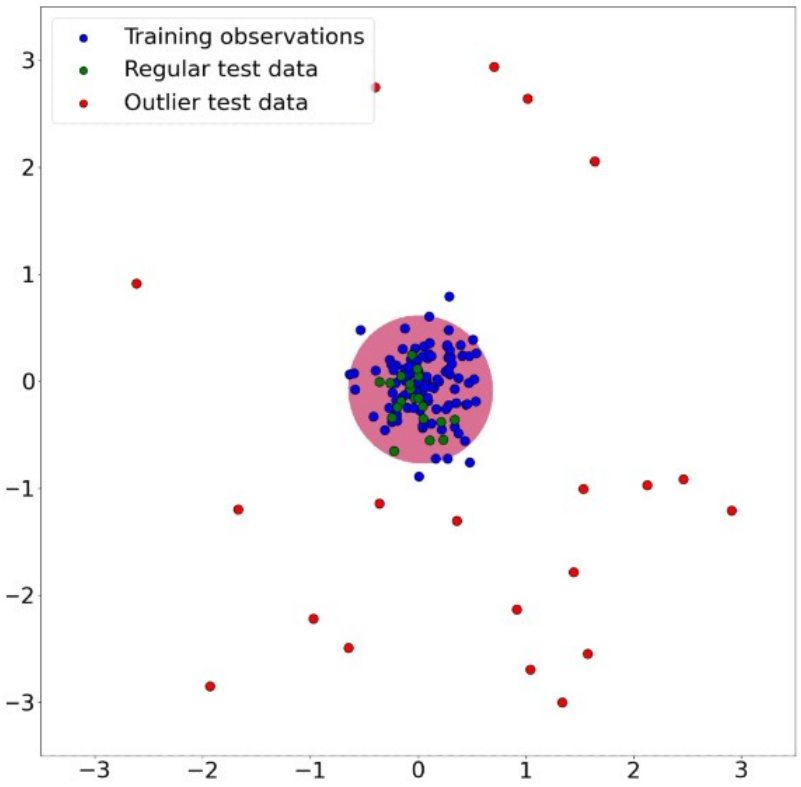

FIGURE 1.4 – Normal Distribution for One-Class SVM.

1.6 Applications

Anomaly detection has significant applications across various domains, and the characteristics of each domain influence the choice of anomaly detection methods. Here are examples of applications in different sectors:

1.6.1 Finance

Anomaly detection plays a pivotal role in the financial sector, especially in detecting fraud, managing risks and ensuring the integrity of financial transactions. This is a comprehensive overview of the main applications used in anomaly detection in the financial domain.

— Fraud Detection: Detecting anomalous patterns in financial transactions to identify potentially fraudulent activities.

— Credit Risk Management: Assessing the risk associated with credit applications and identifying unusual borrowing patterns.

— Algorithmic Trading: Detecting anomalies in market data to optimize trading algorithms and mitigate risks.

— Operational Risk Management: Identifying unusual patterns in operational data to prevent and manage risks in financial operations [16].

1.6.2 Cybersecurity

Anomaly detection is a crucial tool for cybersecurity analysts, constantly monitoring vast amounts of network data for potential cyberattacks. However, traditional anomaly detection systems often face limitations.

These systems flag unusual activity, but often lack context to explain why it's suspicious, leading to wasted effort. Additionally, they can generate false positives, identifying normal activity as an attack. This can overwhelm security analysts with irrelevant notifications, wasting valuable time and resources.[17]

Here's a concrete example:

Scenario: A security alert is triggered because a particular computer transmits a large amount of data in a short period. This could be a sign of a hacker exfiltrating data, but it could also be a legitimate activity like a researcher downloading a large dataset. Security analysts need more information to differentiate between these scenarios. They might need to investigate factors like the destination of the data transfer, the user's typical activity patterns, or the type of data being transferred.

An employee working late downloads a large software update, triggering an anomaly alert due to the unusual data transfer volume at that time. This overwhelms security analysts with non-critical events, diverting their attention away from potentially genuine threats [18].

1.6.3 Healthcare

Despite improvements in healthcare instruments, the presence of medical errors remains a severe challenge. Applying machine learning (ML) and artificial intelligence (AI) algorithms in the healthcare industry helps improve patients' health more efficiently. According to, around 86% of healthcare companies use machine learning and artificial intelligence algorithms. These algorithms help in many ways, such as medical image diagnosis , disease detection/classification medical data analysis [, medical data classification, drug discovery, robot surgery, anomalous reading, etc. Recently, researchers have been interested in detecting abnormal activity in the healthcare industry. An anomaly or outlier is defined as a data instance that does not conform with the remainder of that set of data instances. In the healthcare domain, an anomaly is referred to as an unusual health condition or activity of a patient. A vast number of applications have been developed to detect anomalies in medical data. However, no study has been conducted to find out why these points are considered as an anomaly, i.e., on which set of features a data point is dramatically different than others, as far as we know. The problem of detecting such an explanation leads to outlying aspect mining (a.k.a, outlier explanation, outlier interpretation, outlying subspaces detection). Outlying aspect mining aims to identify the set of features where the given point (or a given anomaly) is most inconsistent with the rest of the data [19].

At an abstract level, an anomaly is defined as a pattern that does not conform to expected normal behavior. A straightforward anomaly detection approach, therefore, is to define a region representing normal behavior and declare any observation in the data that does not belong to this normal region as an anomaly. But several factors make this simple approach very challenging.

Defining a normal region that encompasses every possible normal behavior is very difficult. In addition, the boundary between normal and anomalous behavior is often not precise. Thus an anomalous observation that lies close to the boundary can be normal, and vice-versa.

When anomalies are the result of malicious actions, the malicious adversaries often adapt themselves to make the anomalous observations appear normal, thereby making the task of defining normal behavior more difficult.

In many domains normal behavior keeps evolving and a current notion of normal behavior might not be sufficiently representative in the future. —The exact notion of an anomaly is different for different application domains. For example, in the medical domain, a small deviation from normal (e.g., fluctuations in body temperature) might be an anomaly, while a similar deviation in the stock market domain (e.g., fluctuations in the value of a stock) might be considered as normal. Thus applying a technique developed in one domain to another is not straightforward.

Availability of labeled data for training/validation of models used by anomaly detection techniques is usually a major issue.

Often the data contains noise which tends to be similar to the actual anomalies and hence is difficult to distinguish and remove. Due to the above challenges, the anomaly detection problem, in its most general form, is not easy to solve. In fact, most of the existing anomaly detection techniques solve a specific formulation of the problem. The formulation is induced by various factors such as the nature of the data, availability of labeled data, type of anomalies to be detected, etc. Often, these factors are determined by the application domain in Case Studies [20].

1.7 Current Trends and Innovations

The field of outlier detection is constantly evolving, with researchers developing novel techniques and exploring innovative approaches to identify anomalies in data. Here are some of the emerging trends and advancements shaping the future of outlier detection [21, 22].

1.7.1 Deep Neural Networks (DNNs)

DNNs are being increasingly used for outlier detection due to their ability to learn complex patterns and relationships in high-dimensional data [23]. Convolutional Neural Networks (CNNs) are particularly effective for identifying spatial or temporal anomalies in image and time series data, while Recurrent Neural Networks (RNNs) can handle sequential data like sensor readings or text [24].

These are unsupervised learning models that learn compressed representations of data. Outliers are often poorly reconstructed by the autoencoder, making them detectable through reconstruction errors. Variational Autoencoders (VAEs) further enhance this approach by incorporating probabilistic techniques [25].

a) Function: DNNs are powerful models that can learn complex, non-linear relationships within data ,This allows them to identify patterns that deviate significantly from "normal" data points, potentially indicating anomalies [26].

b) Advantages:

— Highly effective in high-dimensional data: Healthcare data often involves numerous features (e.g., blood pressure, lab tests, medications). DNNs excel at handling such complexity [27].

— Flexibility: DNNs can be adapted for various data types - images (X-rays, CT scans), time series (vital signs), or even text (medical records). Specific

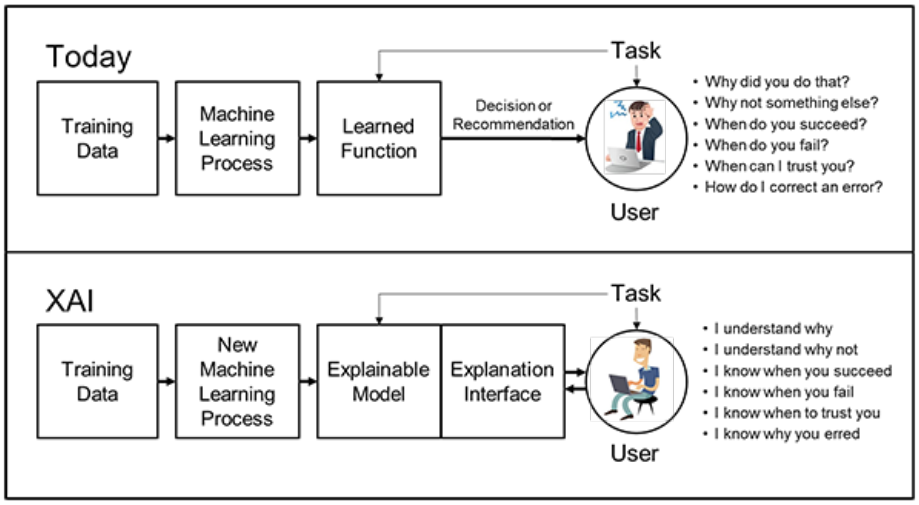

FIGURE 1.5 – XAI Concept [33].

network architectures like Convolutional Neural Networks (CNNs) for images or Recurrent Neural Networks (RNNs) for sequential data can further enhance performance [28].

c) Limitations:

— Black Box Problem: DNNs can be difficult to interpret. Healthcare professionals might struggle to understand why a specific data point is flagged as an anomaly. This lack of explainability can hinder trust and adoption [29].

— Data Dependency: DNNs require large amounts of high-quality training data to function effectively In healthcare, where data privacy is paramount, acquiring sufficient labeled data can be challenging [30].

1.7.2 Explainable Outlier Detection

Traditionally, outlier detection algorithms often lacked transparency, making it difficult to understand why specific data points were flagged as anomalies. This is being addressed by research on explainable AI (XAI) techniques, aiming to provide interpretable insights into the reasoning behind outlier detection decisions [31].

Explainable Artificial Intelligence (XAI), also known as Interpretable AI or Explainable Machine Learning (XAI), refers to a set of methods and tools that aim to make artificial intelligence systems, particularly machine learning algorithms, understandable and interpretable by humans, Machine learning models are often seen as opaque "black boxes" whose decisions are difficult to understand . XAI seeks to lift this veil by explaining the reasoning behind the predictions and actions of AI systems [32]. Figure 1.5 shows the XAI Concept.

1.7.3 Active Learning for Outlier Detection

Active learning in outlier detection introduces an interactive learning element. The model iteratively queries the user for labels on specific data points. This allows the model to focus its learning on the most informative data points. Here's how it works [34]. The model analyzes the data and identifies data points that are most

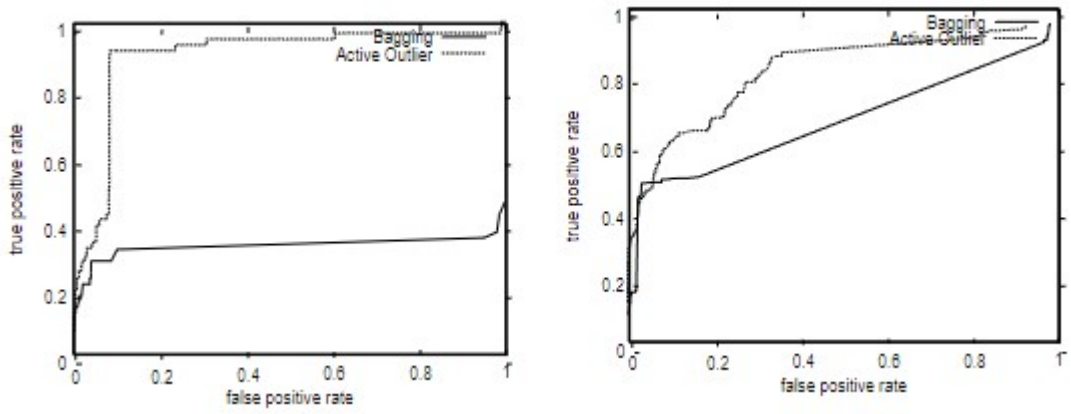

FIGURE 1.6 – Active Outlier and Bagging for KDD99 and Mammography [36].

uncertain - those closest to the decision boundary between normal and anomalous data. The model then queries the user for labels on these uncertain points.

By incorporating the user's labels, the model refines its understanding of the data distribution and becomes more adept at identifying outliers with fewer labeled samples.

This approach is particularly valuable when dealing with large datasets where labeling all data points can be expensive or time-consuming. However, it requires user expertise to provide accurate labels for the queried points [35].

An illustrative example is given by Figure 1.6.

1.7.4 Context-aware Outlier Detection

Real-world data often has inherent context, such as seasonal variations, sensor locations, or even user demographics. Traditional outlier detection methods might struggle to distinguish between true anomalies and expected variations within this context.

Context-aware outlier detection techniques address this challenge by incorporating contextual information into the model. By considering context, the model can:

— Improve anomaly detection accuracy: It can differentiate between true outliers and data points that deviate from the norm due to contextual factors.

— Reduce false positives: By understanding expected contextual variations, the model avoids flagging normal data points as anomalies. Here are some examples of how context can be incorporated:

— Time-series data: The model can account for seasonal trends or cyclical patterns to avoid flagging normal fluctuations as outliers.

— Sensor data: The model can consider sensor location and expected variations based on that location.

— User data: The model can take into account user demographics or historical behavior patterns to identify unusual activity specific to that user [37].

1.7.5 Federated Learning for Outlier Detection

Federated learning is an emerging technique that enables collaborative learning across multiple devices or data centers while preserving data privacy . This is particularly useful in scenarios where data is distributed across different locations, and data privacy is a major concern, such as in healthcare or finance.

Here's how federated learning can be applied to outlier detection:

Each device or data center trains a local model on its own private data.

The models then exchange information (parameters or gradients) without revealing the underlying data itself.

By aggregating this information, a global model is built that can effectively detect outliers across the entire distributed dataset while maintaining data privacy.

Federated learning holds promise for outlier detection in various domains where data is distributed and privacy is paramount. However, it presents challenges like communication overhead and ensuring convergence of the global model with local updates [38].

1.8 Conclusion

This chapter has comprehensively explored the world of anomaly detection, a vital tool for identifying the unusual across diverse domains. We established a strong foundation, examining different types of anomalies and the challenges associated with their detection. We then explored traditional methods and delved into the power of machine learning techniques, showcasing their strengths and applications in the real world.

However, anomaly detection is just one piece of the puzzle. While it excels at identifying outliers, a deeper understanding of data requires exploring the underlying structure and relationships between elements. This is where word embedding techniques, the focus of the next chapter, come into play.

Word embedding techniques provide powerful tools for capturing the semantic relationships between words and concepts within data. By analyzing data through this lens, we can go beyond simply identifying anomalies. We can uncover hidden patterns, understand the relationships between different data points, and gain a richer, more comprehensive understanding of the information at hand.

By combining the power of anomaly detection with the insights offered by word embeddings, we gain a more complete picture of our data. We can identify outliers, understand their context, and delve deeper into the relationships and structures that exist within the data itself. This combined approach paves the way for enhanced decision-making capabilities and ultimately leads to a richer understanding of the world around us.

Chapter 2

Word Embedding Techniques

2.1 Introduction

In the previous chapter, we explored the fascinating world of outlier detection – identifying unusual data points that don't fit the mold. But how can we truly understand the "meaning" of these outliers if they're just numbers or strange characters? This is where Natural Language Processing (NLP) steps in, specifically the concept of word embeddings. Imagine turning words into numerical codes, like fingerprints that capture their essence, relationships, and even grammatical roles. These codes, far from being random, reveal hidden connections between words. Now, think of outlier detection not just for numbers, but for words that seem out of place in a text. Word embeddings can help us understand why a word might be an outlier – is it a grammatical oddity, a completely unrelated concept, or perhaps a hint of hidden sarcasm? This chapter dives deep into word embedding techniques, unlocking the secrets of how computers can not only understand the meaning behind words, but also use that knowledge to identify outliers that stand out from the crowd. By exploring the hidden dimensions of language, we can push the boundaries of NLP and make sense of even the most unexpected words.

2.2 Definitions

2.2.1 Definition 1

Word embedding refers to a set of machine learning techniques that aim to represent the words or phrases of a text by vectors of real numbers, described in a vector model (or Vector Space Model). These new textual data representations have improved the performance of automatic language processing (or Natural Language Processing) methods, such as Topic Modeling or Sentiment Analysis [39].

Word embedding is based on the linguistic theory founded by Zelling Harris and known as Distributional Semantics. This theory considers that a word is characterized by its context, ie by the words that surround it. Thus, words that share similar contexts also share similar meanings. Word embedding algorithms are most often used to describe words through digital vectors, but they can also be used to build vector representations of whole sentences, biological data such as DNA sequences, or networks represented as graphs [39].

2.2.2 Definition 2

Word embedding in NLP is an important term that is used for representing words for text analysis in the form of real-valued vectors. It is an advancement in NLP that has improved the ability of computers to understand text-based content in a better

way. It is considered one of the most significant breakthroughs of deep learning for solving challenging natural language processing problems.

In this approach, words and documents are represented in the form of numeric vectors allowing similar words to have similar vector representations. The extracted features are fed into a machine learning model so as to work with text data and preserve the semantic and syntactic information. This information once received in its converted form is used by NLP algorithms that easily digest these learned representations and process textual information [40].

2.3 Traditional Methods vs. Word Embedding

2.3.1 Word Embeddings

a) Concept: Word embeddings represent words as vectors in a high-dimensional space. Words with similar meanings are positioned closer in this space. This is achieved using machine learning techniques like Word2Vec or GloVe that analyze large text corpora to capture semantic relationships.

b) Strengths: Captures semantic similarities between words. Enables tasks like finding synonyms or analogies. Performs well on complex NLP tasks .

c) Weaknesses: Computationally expensive to train compared to BoW or TF-IDF. Requires a large amount of training data [41].

2.3.2 TF-IDF

a) Concept: TF-IDF (Term Frequency-Inverse Document Frequency) builds on BoW by addressing some of its limitations. It considers both the word frequency (TF) within a document and its inverse document frequency (IDF) across the entire corpus. Words that appear frequently in a single document but rarely overall get a higher TF-IDF weight .

b) Strengths: Improves on BoW by weighting words based on their importance. Helps identify keywords specific to a document .

c) Weaknesses: Still doesn't capture semantic relationships between words [41].

Figure 2.1 shows an illustrative example.

2.3.3 Bag-of-Words (BoW)

a) Concept: BoW treats a document as a "bag" of words, ignoring the order and grammar. It creates a feature vector where each element represents a word and its value indicates the word's frequency in the document.

b) Strengths: Simple and efficient to implement. Works well for small datasets .

c) Weaknesses: Ignores word order and relationships between words. Doesn't capture semantic similarities (e.g., "king" and "queen") [42].

A comparison between Word Embeddings, BoW, and TF-IDF is given in Table 2.1.

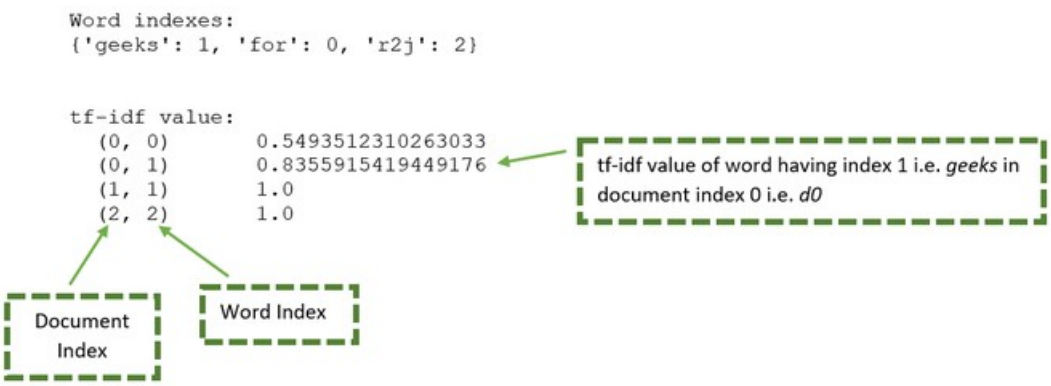

FIGURE 2.1 – The resultant variable comprises unique words along with their corresponding TF-IDF values.

TABLE 2.1 – Comparison of Word Embeddings, BoW, and TF-IDF [42].

Feature	**BoW**	**TF-IDF**	**Word Embeddings**
Word Order	Ignored	Ignored	Considered
Semantic Similarity	Not captured	Not captured	Captured
Computational Cost	Low	Low	High
Data Requirements	Low	Moderate	High

2.4 Key Concepts in Word Embedding

2.4.1 Semantic Similarity

Measuring the semantic similarity of texts has a vital role in various tasks from the field of natural language processing (NLP) such as document classification, information retrieval , word sense disambiguation, plagiarism detection, etc. The specific task of measuring the semantic similarity of short texts is of importance in the domain of social media for opinion mining, recommendation, event detection, and news recommendation Representing short texts may differ from representing long texts due to the sparsity and noisines.

Hence, it is important to develop approaches focused only on short texts such as tweets, comments, or microblogs, Therefore, approaches that are tailored to short texts may not work well with long texts and vice versa, Firstly, word embeddings are derived for one word.

In the case of short texts, it is necessary to scale up from word embeddings to text embedding. A large number of techniques that leverage this issue are proposed, and still, there is no consensus in the research community on how to proceed. One possibility is to take the sum or the average (centroid) of the individual word embeddings for all the words in the text. This approach has been widely adopted in many studies "Sentence Similarity Techniques for Short vs Variable Length Text using Word Embeddings, Measuring Semantic Similarity of Short Texts: An Experimental Study, Short Text Similarity with Word Embeddings", for example, and in general, they perform well. However, by aggregating a set of word embeddings into only one embedding as (averaged or weighted) sum or centroid, we are losing valuable semantic information. This happens because of reducing the information contained in the set of vectors into one vector [43].

FIGURE 2.2 – Example for Semantic Space Visualizations [45].

2.4.2 Word Vectors and Semantic Spaces

Most current word embedding algorithms build on the distributional hypothesis where similar contexts imply similar meanings so as to tie co-occurrences of words to their underlying meanings. The relationship between semantics and cooccurrences has also been studied in psychometrics and cognitive science , often by means of free word association tasks and semantic spaces. The semantic spaces, in particular, provide a natural conceptual framework for continuous representations of words as vector spaces where semantically related words are close to each other. For example, the observation that word embeddings can solve analogies using vector representations of words derived from surveys of pairwise word similarity judgments [44]. An example is given in Figure 2.2.

2.5 Popular Word Embedding Models

2.5.1 Word2Vec

Word2Vec, a powerful technique in natural language processing (NLP), has gained significant attention in recent years for its ability to capture semantic meaning in words. By training on large text corpora, word2vec generates vector representations for words, where words with similar meanings reside close together in this vector space. This allows applications like sentiment analysis, recommendation systems, and machine translation to leverage the semantic relationships between words [46]. Figure 2.3 shows an example.

a) Continuous-bag-of-words architecture: The core idea behind CBOW is that words appearing close together in a sentence likely share similar meanings. The objective is to predict a center word c given its surrounding words w_i as shown in Equation 2.1.

$$P(c \mid w_1) \times P(c \mid w2) \times \cdots \times P(c \mid w_n) \tag{2.1}$$

This represents the probability of the center word c appearing considering each surrounding word w_i (where i goes from 1 to n) individually. We want to maximize this probability (P) for the model to perform well [48].

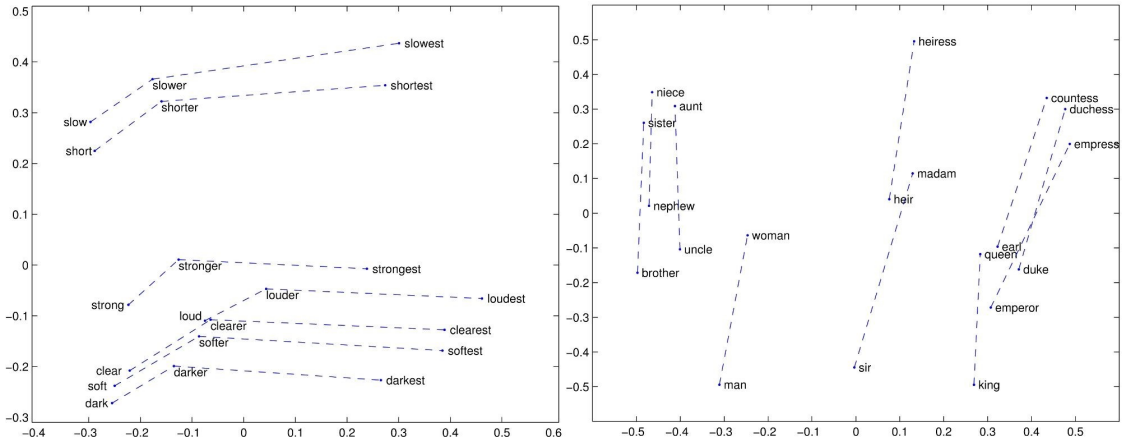

FIGURE 2.3 – Example of how Word2Vec works [47].

b) Skip-gram: Unlike CBOW, Skip-gram takes the opposite approach. As shown in Equation 2.2, instead of predicting the center word, it aims to predict surrounding words based on a center word.

$$\sum_{t=1}^{T} \sum_{\substack{c \leq j \leq c \\ j \neq 0}} \log(P(w_{t+j} \mid w_c)) \tag{2.2}$$

Where:

T: number of sentences in the training text.
t: index iterating over each sentence.
c: window size (context words considered around the center word).
j: index iterating over surrounding words within the window.
$logP(w_{t+j} \mid w_c)$: This calculates the sum of the log probabilities of predicting each surrounding word w_{t+j} (where j goes from $-c$ to $+c$, excluding the center word w_c) given the center word w_t. Essentially, we maximize the likelihood of predicting nearby words accurately.
Softmax function: This function (Equation 2.3) ensures the probabilities of all surrounding words sum up to 1.

$$P(w_O \mid w_I) = \frac{\exp\left(v'_{w_O} \cdot v_{w_I}\right)}{\sum_{w=1}^{W} \exp\left(v'_w \cdot v_{w_I}\right)} \tag{2.3}$$

Where:

$P(w_0|w_I)$: This represents the probability of predicting the surrounding word w_O given the center word w_I.

$v'w_O$, vw_I: These represent the vector representations of words w_O and w_I, respectively.

$\sum^W$: summation over all words in the vocabulary.

exp(): exponential function [48].

2.5.2 GloVe (Global Vectors for Word Representation)

The statistics of word occurrences in a corpus is the primary source of information available to all unsupervised methods for learning word representations, and although many such methods now exist, the question still remains as to how meaning is generated from these statistics, and how the resulting word vectors might represent that meaning. In this section, we shed some light on this question. We use our insights to construct a new model for word representation which we call GloVe, for Global Vectors, because the global corpus statistics are captured directly by the model, unlike other word embedding models that focus on local co-occurrence (how often words appear together in a window), GloVe leverages statistical information from the entire corpus. It does this by focusing on the ratio of co-occurrence probabilities between words [49]. Here's the key idea:Imagine words A, B, and C.

If A and B frequently co-occur (appear together), and B also co-occurs with C, but not as often as with A, then intuitively, A and C should share some semantic similarity (meaning) [49].

Generating the raw word co-occurrence counts is simply a matter of going through a large spoken or written corpus and counting the number of times n(c,t) each context word c occurs within a window of a certain size W around each target word t. We shall assume that the corpus is used in its raw state, with no preprocessing, thus giving us a conservative estimate of the performance levels achievable. Humans may well make use of simple transformations, such as stemming or lemmatization , as they experience the stream of words, and thus form better representations than our basic counting approach. For example, they might improve their performance by making use of the kind of grammatical knowledge that tells us that "walk" and "walked" are morphologically and thus semantically related [50].

2.5.3 FastText

It extends traditional word embeddings, which represent entire words as vectors, by incorporating sub-word information [19]. It achieves this by decomposing words into character n-grams (subword sequences of length n) and learning embeddings for these n-grams [20]. These n-gram embeddings are then summed to represent the entire word [51].

a) Advantages for Morphologically Rich Languages: Handling Out-of-Vocabulary (OOV) Words: Morphologically rich languages often have complex word formation processes, leading to frequent creation of new words (OOV words) not present in the training vocabulary. By using sub-word information, FastText can represent OOV words by combining the embeddings of their constituent n-grams, even if the entire word itself is not encountered during training [52].

Improved Representation of Morphologically Related Words: Since sub-words (n-grams) capture common morphemes (meaningful units) across words, FastText can capture semantic relationships between words that share these morphemes, even if they differ in prefixes or suffixes.

b) Word as Sum of n-gram Embeddings: The fundamental idea behind FastText's word representation is capturing a word's meaning by summing the embeddings of its constituent n-grams. Let's denote:

w as a word.

n as the n-gram size (length of character subsequences).

$C(w)$ as the set of all n-grams in word w (e.g., bigrams for n=2).

$E(x)$ as the embedding vector for element x (word or n-gram).

The embedding vector for word w, denoted as $E(w)$, can be expressed as the sum of the embedding vectors of its n-grams in $C(w)$ as given by Equation 2.4.

$$E(w) = \sum_{ng \in C(w)} E(ng) \tag{2.4}$$

This formula essentially states that the word embedding is obtained by summing the individual n-gram embeddings that make up the word [53].

2.5.4 BERT: Bidirectional Encoder Representations from Transformers

BERT stands as a landmark achievement in Natural Language Processing (NLP). Its success hinges on two key elements: the transformer architecture and bidirectional context modeling. Let's delve into these concepts and explore how BERT has revolutionized contextualized word embeddings.

At the heart of BERT lies the transformer architecture. Introduced in the paper "Attention is All You Need" by Vaswani et al. (2017), the transformer departs from traditional recurrent neural networks (RNNs) for sequence processing. Here's a simplified breakdown:

a) Encoder-Decoder Structure: The transformer utilizes an encoder-decoder structure. The encoder processes the input sequence, capturing its relationships and dependencies. The decoder, used in some applications like machine translation, generates an output sequence based on the encoded information.

b) Attention Mechanism: The core strength of the transformer lies in its attention mechanism. This mechanism allows the model to focus on specific parts of the input sequence that are most relevant for the current processing step. Unlike RNNs, which process information sequentially, the attention mechanism enables a more parallel and efficient approach [54].
Attention Score: (Equation 2.5).

$$\text{Attention}(X_i, X_j) = \text{softmax}\left(\frac{Q_i \cdot K_j^T}{d_k}\right) \tag{2.5}$$

- X_i: Embedding of the i-th token in the sequence.
- X_j: Embedding of the j-th token in the sequence.
- Q_i: Query vector for the i-th token (obtained by multiplying the embedding X_i with the query matrix Q).
- K_j^T: Transpose of the key vector for the j-th token (obtained by multiplying the embedding X_j with the key matrix K).
- d_k: Dimensionality of the key vectors (number of features in each key vector).
- softmax(x): Softmax function, which transforms a vector of real numbers into a

probability distribution over the elements.

Attention Matrix: This formula isn't explicitly shown, but it's essentially applying the attention score calculation between all possible pairs of tokens (i, j) in the sequence. The resulting matrix will have dimensions (number of tokens) × (number of tokens), where each cell (i, j) represents the attention score between the i-th and j-th tokens.[55]

2.6 Training and Fine-Tuning Word Embedding

2.6.1 Training Word Embeddings

Several techniques exist for training word embeddings, each with its own advantages and limitations. Here are two common approaches:

a) Statistical Methods: Word2Vec This popular method, introduced by Mikolov et al. (2013) [https://arxiv.org/abs/1301.3781] utilizes two architectures. Continuous Bag-of-Words (CBOW): Predicts a target word based on its surrounding context words. Skip-gram: Predicts surrounding context words based on a given target word. GloVe: Developed by Pennington et al. (2014) [https://nlp.stanford.edu/pubs/glove.pdf], GloVe leverages the co-occurrence statistics of words within a large corpus to learn word embeddings.

b) Neural Network-based Methods: Word2Vec Can also be implemented using neural networks, offering more flexibility and potentially better performance compared to the statistical approach. FastText: Introduced by Joulin et al. (2016), extends Word2Vec by considering subword information (character n-grams) to handle out-of-vocabulary words and improve representation for morphologically rich languages [24].

Adjusting various parameters such as the context size, vector size, etc., is essential in training both Word2Vec and FastText models. The context size parameter determines the number of surrounding words considered when predicting the target word, while the vector size parameter determines the dimensionality of the learned word embeddings. Other parameters, such as learning rate, batch size, and training epochs, also play crucial roles in model training and performance. optimizing these parameters requires experimentation and tuning to achieve the desired performance for specific tasks and datasets. Techniques such as grid search or random search can be employed to systematically explore the parameter space and identify optimal configurations.[58]

2.6.2 Fine-Tuning Pre-trained Embeddings

While training word embeddings from scratch can be effective, leveraging pre-trained embeddings often offers significant advantages, especially for tasks with limited training data [25]. These pre-trained embeddings are typically obtained from large-scale language models (LLMs) [26] trained on massive text corpora. However, these generic embeddings might not be optimal for specific NLP tasks.

Fine-tuning techniques aim to adapt these pre-trained embeddings to a particular task:

1) Freezing Layers:

This technique focuses on leveraging the general linguistic knowledge captured in

the lower layers of the pre-trained model, which encode general word meanings and relationships [60]. These layers are frozen, meaning their weights are not updated during training. Conversely, the higher layers, responsible for more task-specific features, are trained on the new task data, allowing them to adapt to the specific requirements of the task.

a) The Challenge:
Pre-trained word embedding models consist of multiple layers, each layer building upon the previous one to learn increasingly complex representations of language.

Lower Layers: These capture fundamental word meanings and relationships, forming a strong foundation for language understanding.

Higher Layers: These layers process the information from lower layers and learn more task-specific features relevant to the pre-training objective (e.g., predicting surrounding words, sentence similarity).

The challenge lies in adapting a pre-trained model to a new NLP task while preserving this valuable knowledge.

b) The Solution:
Freezing layers is a technique used during fine-tuning that selectively updates the weights of different layers in the pre-trained model:

Freezing Lower Layers: The weights of the lower layers, responsible for capturing general word meanings and relationships, are frozen. This means their values remain unchanged during training on the new task data.

Training Higher Layers: The weights of the higher layers, which learned task-specific features for the pre-training objective, are free to be updated during training. These layers will adapt to the specific features relevant to your new NLP task [61].

2) Reduced Learning Rate: This method aims to fine-tune the entire pre-trained model, but with smaller adjustments to the weights compared to training from scratch. This is achieved by using a lower learning rate during training. This prevents the model from completely forgetting the general knowledge learned during pre-training while allowing it to specialize for the specific task at hand [62].

a) The Challenge: Pre-trained word embeddings capture a vast amount of general knowledge about language. However, directly applying them to a specific NLP task might not be optimal. The model needs to adapt these pre-trained embeddings to the nuances of your particular task without losing the valuable knowledge they already contain.

b) The Solution: Reduced learning rate is a technique used during fine-tuning to achieve this balance. Here's how it works: Fine-tuning the Pre-trained Model: We take a pre-trained word embedding model (like Word2Vec or GloVe) that has already learned general word relationships. We use this model as a starting point for our specific NLP task. This is called fine-tuning because we are adjusting the pre-trained model for a new task.

Learning Rate and Weight Adjustments: During training, the model updates the weights of its internal connections to improve its performance on the new task. The learning rate controls the magnitude of these weight updates. A higher learning rate leads to larger adjustments, while a lower learning rate results in smaller, more gradual changes.

Reduced Learning Rate in Action: By using a reduced learning rate, we make smaller adjustments to the weights of the pre-trained word embeddings. This prevents the model from drastically altering the valuable knowledge it learned during pre-training. Imagine the pre-trained embedding for "happy" is associated with

words like "joyful" and "excited." With a high learning rate, the model might completely overwrite these connections if the new task emphasizes a different aspect of "happy". A reduced learning rate allows the model to refine these connections while preserving the core understanding of "happy". It might learn to associate it with words like "elated" or "delighted" in the context of the specific task [63].

3) Task-specific Data Augmentation: This technique recognizes that the generic pre-trained embeddings might not be optimal for the specific task. It suggests enriching the training data with techniques like back-translation (generating translations and then translating them back to the original language) or other data augmentation methods specific to the target task. This helps the model learn more relevant representations for the task data [64].

a) The Challenge: Pre-trained word embeddings, while powerful, might not be perfect for every NLP task. They capture general word relationships based on the massive datasets they are trained on . These broad relationships might not capture the nuances specific to your particular task.

b) The Solution: Task-specific data augmentation aims to enrich your training data and improve the word embeddings for your specific NLP task. Here's how it works:

c) Leveraging Existing Techniques: Back-Translation: This method involves translating your existing training data from your target language (let's say English) to another language (like French) and then translating it back to English . This process can introduce variations in sentence structure and word choice, effectively augmenting your data. Impact on Embeddings: When the model is trained on both the original and back-translated data, the word embeddings get exposed to these variations. They learn to capture the nuances of meaning that might be present in different phrasings of the same concept . For example, the embedding for "happy" might be strengthened in its association with words like "elated" or "delighted" if those appeared in the back-translated data [65].

2.7 Applications of Word Embedding

Word embeddings, numerical representations of words, have revolutionized various Natural Language Processing (NLP) tasks. Their ability to capture semantic relationships between words makes them valuable tools, improving performance in diverse areas like sentiment analysis, named entity recognition, and machine translation [66]. Let's delve into how word embeddings enhance these specific tasks:

2.7.1 Sentiment Analysis

Sentiment analysis aims to determine the sentiment (positive, negative, or neutral) expressed in a text piece [67]. Word embeddings assist by:

Capturing sentiment-oriented information: Words with similar sentiment vectors tend to express similar emotions. For example, "happy" and "joyful" will have closer vectors compared to "sad" or "angry." By analyzing the average sentiment vector of words in a sentence, the model can infer the overall sentiment [68].

Handling out-of-vocabulary (OOV) words: Traditional methods might struggle with OOV words (words not present in the training data). Word embeddings, especially those utilizing subword information (e.g., FastText), can represent OOV words by combining the embeddings of their constituent subwords [69].

How Word Embeddings help:

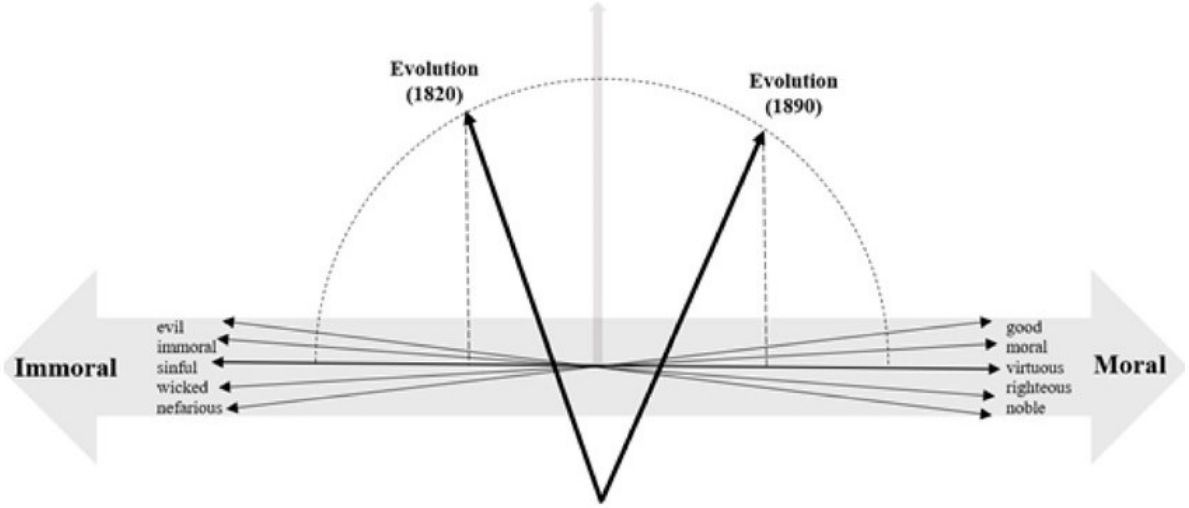

FIGURE 2.4 – Sentiment Analysis with Embedded Vectors [70].

a) Capture Sentiment-oriented Information: Word embeddings learn relationships between words. Words with similar sentiment tend to have closer vectors in embedding space. Example: "happy" and "joyful" will have embeddings closer to each other than "sad" and "angry".

b) Handle Out-of-Vocabulary (OOV) Words: Traditional methods might struggle with words not present in the training data (OOV words). Word embeddings, especially those using subword information (e.g., FastText), can represent OOV words by combining the embeddings of their parts (subwords).

Example:

Sentence 1: "The movie was fantastic!" (Positive sentiment)

Sentence 2: "The film was terrible." (Negative sentiment)

2.7.2 Named Entity Recognition

NER identifies and classifies named entities (e.g., people, locations, organizations) within a text [34]. Word embeddings contribute to improved NER performance by:

Encoding contextual information: Word embeddings capture the relationships between words, which is crucial for NER. For instance, the word "Paris" might have a different embedding depending on the context (e.g., "visited Paris" vs. "the city of Paris"). This context-dependence helps in identifying entities accurately [71].

Encoding contextual information: Word embeddings capture the relationships between words, which is crucial for NER. For instance, the word "Paris" might have a different embedding depending on the context (e.g., "visited Paris" vs. "the city of Paris"). This context-dependence helps in identifying entities accurately [72].

Example:

Sentence 1: "I flew to Paris for the weekend."

Sentence 2: "That hat is very Parisian."

"Paris" as a location in Sentence 1.

"Parisian" as an adjective derived from the location in Sentence 2.

The model considers the surrounding words ("flew to") to understand "Paris" refers to a location in Sentence 1.

In Sentence 2, "Parisian" modifies the noun "hat," indicating it's not a location.

Word embeddings are not static. The model can adjust a word's embedding based on its context.

"Paris" in Sentence 1 will have a different embedding compared to "Parisian" in Sentence 2 [73].

2.7.3 Machine Translation

Machine translation involves automatically translating text from one language to another [36]. Word embeddings play a vital role by:

a) Bridging the semantic gap between languages: By learning the relationships between words within each language, word embeddings can bridge the gap between semantically similar words in different languages. This allows the model to find the most appropriate translations that preserve the meaning of the original text [74].

b) Handling complex sentence structures: Word embeddings can capture the syntactic relationships between words, which is crucial for translating complex sentence structures accurately [75].

Example:

Source Sentence (English): "The bank is on the river".

Desired Translation (French): "La banque est sur la rivière". (The bank is on the river)

Understanding "Bank" in Context: The model uses word embeddings to understand that "bank" in this context refers to a financial institution, not the edge of a river.

Finding Similar Meaning in French: The model searches for a French word with a similar embedding to the English "bank" in a financial context.

Accurate Translation: By identifying "banque" (bank) as the closest semantic equivalent, the model produces the desired translation.

2.8 Evaluation of Word Embeddings

Evaluating the quality of word embeddings is crucial in assessing their effectiveness for NLP tasks, Several common metrics and benchmark datasets are used for this purpose, but the process is not without its challenges [76].

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.6}$$

$$Precision = \frac{TP}{TP + FN} \tag{2.7}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.8}$$

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{2.9}$$

2.8.1 Evaluation Metrics

1. Analogy Tasks: These tasks evaluate the ability of embeddings to capture semantic relationships between words. Examples include:

• Word similarity: Measuring the distance between embedding vectors of words with similar meanings (e.g., "king" and "queen").

• Word analogy: Assessing if the relationship between two word pairs holds for their embeddings (e.g., "man" is to "woman" as "king" is to "queen").[77]

2. Intrinsic Evaluation: This approach directly evaluates the quality of the embeddings themselves, focusing on properties like: Cosine similarity:

• Measuring the similarity between embedding vectors using the cosine distance metric.

• Clustering: Evaluating if words with similar meanings are grouped together when their embeddings are clustered. [78]

2.8.2 Challenges

1. Lack of Ground Truth: There's no single universally accepted measure of "good" word embeddings, making it challenging to definitively evaluate their quality.

2. Task-Dependence: The effectiveness of word embeddings can vary significantly depending on the specific NLP task they're used for. Evaluating them in isolation might not always reflect their true performance in real-world applications. [79]

3. Interpretability: Understanding the exact information encoded in word embeddings can be difficult, making it challenging to pinpoint specific strengths or weaknesses and explain their performance on specific tasks.

2.9 Conclusion

In this chapter, we embarked on a journey through the intricate landscape of outlier detection, uncovering its significance across diverse domains. We began by introducing the fundamental concept of outlier detection and underscored its pivotal role in identifying anomalies within data, crucial for maintaining data integrity and making informed decisions.

We then traversed through the taxonomy of outliers, delineating various types such as point anomalies, contextual anomalies, and collective anomalies, each presenting unique challenges and intricacies. Traditional methods, including statistical approaches like z-score and rule-based techniques, provided a solid foundation for understanding outlier detection's historical context.

As we delved deeper, we explored the importance of evaluation metrics such as precision, recall, F1-score, and AUC-ROC, essential for assessing the efficacy of outlier detection algorithms and guiding their deployment in real-world scenarios.

Real-world applications spanning finance, healthcare, and cybersecurity illuminated the practical relevance of outlier detection, showcasing its instrumental role in fraud detection, anomaly monitoring in patient health data, and identifying malicious activities in network traffic. Despite the strides made, we confronted persistent challenges, including imbalanced datasets and interpretability issues, underscoring the ongoing need for innovative solutions and robust methodologies.

Through compelling case studies, we witnessed the tangible impact of outlier detection, unveiling valuable insights and driving actionable decisions across diverse domains. As we concluded our exploration, we navigated through current trends and innovations, heralding a future ripe with possibilities. The chapter's essence lies in the acknowledgment of outlier detection's dynamic nature, urging practitioners to adapt and innovate in response to evolving data landscapes and emerging challenges. In essence, outlier detection stands as a cornerstone in the realm of data analysis, guiding us towards a deeper understanding of our data and empowering us to extract meaningful insights amidst the noise. As we forge ahead, the importance of selecting appropriate methods attuned to the intricacies of each application remains paramount, ensuring outlier detection continues to serve as a beacon of clarity amidst the complexities of modern data analytics.

Chapter 3

Conceptual Framework

3.1 Introduction

This chapter explains how to evaluate similarity and detect outliers, aiming to develop new algorithms in this area. Our goal is to combine high-level ideas and details to create a clear framework for understanding similarity evaluation and outlier detection. We will explore techniques like Word Embeddings Models to use advanced methods for improving algorithms.

In this study, we will use advanced text processing techniques to detect outlier documents. First, we will clean and organize the text data. Then, we will use word representation models called embeddings to understand the meaning of words in a continuous vector space. With these embeddings, we will calculate "Word Mover's Distance" (WMD) to compare the semantic similarity between documents. This approach will help us identify documents that differ significantly in meaning, marking them as outliers.

3.2 Theoretical Foundations

The ideas of similarity evaluation and outlier detection are motivated and have very deep theoretical ground. Similarity assessment stands for the reclining of the degree of similarity or similarity between data points, which is often essential for tasks such as clustering, classification, or suggestion algorithms. Moreover, outlier detection refers to the identification of data points that work off the presumed or the norm data in several datasets. These techniques provide insight into potential exclusions or faults. While, theoretical basis similarity evaluation often utilizes concepts of mathematics, statistics, and information theory, for instance, functions that measure similarity in different contexts include Euclidean distance, Cosine similarity, Jaccard index. There is a large family of algorithms related to similarity evaluation, and most of them are based on these metrics. Each algorithm may work differently and apply to various data types [80].

In outlier detection, the theoretical foundations are deeply rooted in probability theory, distribution analysis, and anomaly detection techniques. By using statistical principles, methods like z-score, Mahalanobis distance, and density-based approaches can identify data points that exhibit unexpected behavior. To develop outlier detection algorithms that can distinguish genuine anomalies from noise or data artifacts, it is crucial to understand these theoretical constructs [80].

3.2.1 Impact of Outlier Detection on Decision-Making

Outlier detection is both a tool for analyzing information, as well as a mover of decision-making positions inside organizations. In influencing the quality of

information and revealing abnormal instances, it may directly affect others. First, this mechanism promotes belief and reliance on facts through maintenance of soundness as well as trustworthiness in them [109].

As a result, stakeholders are encouraged to base their decisions on accurate information and as a result get better outcomes. Second, when a risk is detected in time by finding an abnormal case it helps in risk management because we can take proactive measures before risks become a problem [81]. Notably, their perceptions and decision-making can be influenced positively if they feel secure and stable through a proactive way of dealing with risks. Besides, innovation is boosted by outlier detection that reveals covert patterns or trends existing within its dataset [82].

The novel information will make the organization come up with its strategies that lead to it earning competitive advantages from different fronts at a go. In addition, detection of outliars makes team members who work together come to a consensus over the dataset and what it really means [83].

3.2.2 Overview of Relevant Literature

An interesting study by Park et al., that tackles the challenge of outlier detection specifically for text data. Text can be tricky because it doesn't always behave like numbers. For example, there might be a lot of words that appear just once or twice, but that doesn't necessarily mean they're strange. Park and their team came up with a clever method called TONMF to sift through these text documents and find the real outliers, the ones that truly stand out from the crowd. They even tested it against other methods and found it worked quite well [84].

"A Framework for Outlier Detection and Robust Clustering of Attributed Graphs": This study goes beyond just textual data and addresses outlier detection in attributed graphs. with a network where documents are connected based on similarities, and some documents within these connections seem out of place. This framework, called ORCA (Outlier detection and Robust Clustering for Attributed graphs), aims to identify both these outliers and group similar documents effectively. It achieves this by combining clustering and anomaly detection techniques in a clever way. The authors highlight the benefits of ORCA like its speed, accuracy, and ability to handle large datasets [85].

"Pattern Recognition and Machine Learning": This seminal text provides a comprehensive exploration of pattern recognition and machine learning techniques, including in-depth discussions on similarity evaluation methods such as clustering, nearest neighbor algorithms, and support vector machines. Bishop explains the theoretical foundations of similarity evaluation while also exploring practical applications in different domains, making it an essential resource for both researchers and practitioners [86].

"Anomaly Detection: A Survey" by Chandola et al., offers a systematic overview of anomaly detection techniques, encompassing both traditional statistical approaches and modern machine learning algorithms. The survey discusses the theoretical foundations of outlier detection while also providing insights into practical applications in fields such as cybersecurity, finance, and healthcare. By synthesizing a vast body of research, the survey provides a valuable reference for understanding the landscape of anomaly detection methods and their applicability in different contexts [87].

3.3 Methodological Approach

Techniques tailored to specific data types and applications are included in methodologies for evaluating similarity and detecting outliers. Quantifying the degree of resemblance or proximity between data points is a common methodology in similarity evaluation, while outlier detection focuses on identifying data points that deviate significantly from the norm.

3.3.1 Contribution to Existing Work

a) Integration of Word Mover's Distance (WMD): Unlike traditional methods that often rely solely on lexical or syntactic features, our work integrates the use of Word Mover's Distance (WMD). WMD considers both the semantic similarity of words and their spatial arrangement, providing a more holistic measure of text similarity [88].

b) Utilization of Word2Vec Embeddings: By leveraging Word2Vec embeddings , our approach captures semantic relationships between words in a continuous vector space. This allows for more effective representation of word semantics compared to traditional approaches.

c) Comprehensive Preprocessing: Our work employs comprehensive text preprocessing techniques, including tokenization, stop word removal, punctuation removal, digit removal, and lemmatization. This ensures that the input text data is clean and uniform, enhancing the effectiveness of subsequent analysis.

d) Performance Evaluation and Outlier Detection: We conduct a thorough performance evaluation of our approach, comparing it with existing methods in terms of accuracy, precision, recall, and F1-score. Additionally, we identify outlier documents where the calculated similarity deviates from human judgment, providing insights into the limitations of semantic similarity measures.

e) Focus on Semantic Similarity: Unlike approaches that primarily rely on lexical or syntactic features, our work focuses on capturing semantic similarity between text documents. This allows for a more nuanced understanding of text similarity, reflecting real-world semantic relationships.

3.3.2 Word Mover's Distance (WMD)

Word Mover's Distance (WMD) is a metric specifically designed to measure the difference between two text documents. Unlike traditional distance metrics that operate on fixed-dimensional vectors, WMD considers the semantic meaning of words and their relationships within the documents. In WMD, each document is represented as a list of words, where each word is treated as a separate entity. The distance between two documents is then calculated as the minimum cumulative distance required to move words from one document to another, taking into account their semantic similarities.

WMD utilizes word embeddings (Figure 3.1), such as Word2Vec or GloVe embeddings, to capture semantic relationships between words. WMD's use of semantic similarity allows for more nuanced measures of document dissimilarity than traditional methods such as Euclidean distance or cosine similarity. WMD can be applied

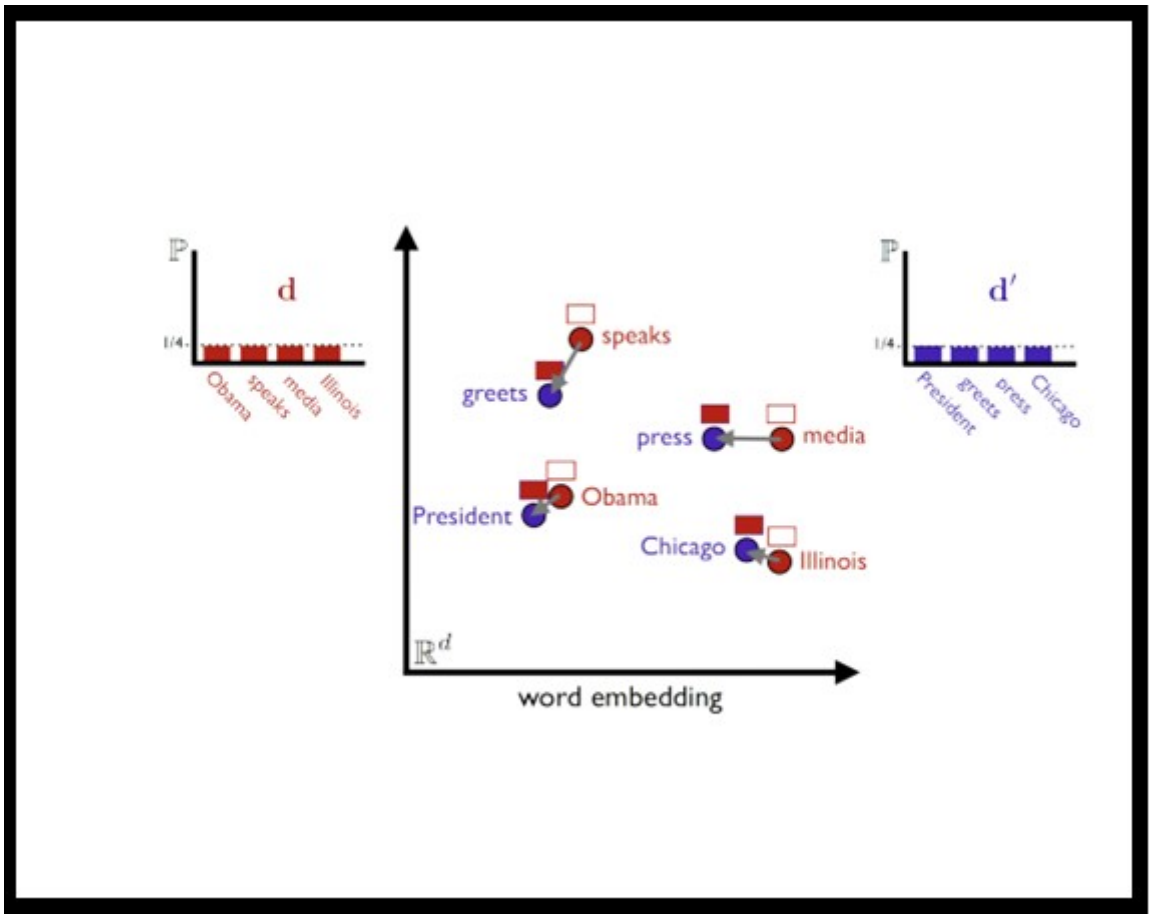

FIGURE 3.1 – The Effect of Word Embedding on Word Mover's Distance [93].

to various natural language processing tasks, such as document clustering, text classification, and information retrieval. Its ability to capture semantic similarities between documents makes it particularly useful in scenarios where the context and meaning of words play a crucial role, such as in sentiment analysis or document summarization [92].

We chose to utilize Word Mover's Distance (WMD) due to its ability to capture semantic similarity between text documents, which is often challenging for classical distance metrics such as Cosine Distance, especially in high-dimensional feature spaces. WMD considers not only the similarity of words but also their spatial arrangement, providing a more nuanced measure of text similarity [94].

Classical distance metrics like Cosine Distance may struggle in high-dimensional spaces due to the "curse of dimensionality", where the effectiveness of distance measures diminishes as the number of dimensions increases [95].

Additionally, Cosine Distance does not account for word semantics and their contextual relationships, which are crucial for capturing the semantic similarity of text documents [96].

By incorporating WMD, we address these limitations and achieve more accurate assessments of text similarity.

3.3.3 Why WMD is better than Cosine Similarity

The original paper which defined 'Word Mover's Distance', "From Word Embeddings To Document Distances" [99], gave some examples of where WMD works well, and comparisons of its behavior against other similarity-calculations.

Table 3.1 compares WMD with Cosine Similarity.

Feature	**Word Mover's Distance (WMD)**	**Cosine Similarity**
Definition	Measures the minimum distance that words from one document need to travel to match words in another document using word embeddings [100].	Measures the cosine of the angle between two non-zero vectors in an inner product space, often used to measure similarity between two documents represented as word vectors [101].
Computation Complexity	Higher computational complexity due to solving an optimization problem (typically $O(n^3 \log n)$ for n words in a document) [102].	Lower computational complexity as it involves basic vector operations ($O(n)$) [103].
Context Sensitivity	Takes the semantic meaning of all words in both documents into account, providing a more holistic similarity measure [100].	Evaluates the similarity based only on the angle between vectors, which might miss nuanced contextual meanings [148].
Handling Synonymy	Effectively handles synonymy as it considers the distance between word embeddings, thus recognizing that different words can have similar meanings [100].	Less effective with synonymy since it relies on exact matches between word vectors rather than their positions in the embedding space [105].
Practical Applications	Useful in applications like document clustering, topic modeling, and semantic search where capturing overall meaning is crucial [100].	Commonly used in applications like information retrieval and text classification where speed and efficiency are important [101].
Empirical Performance	Often shows superior performance in benchmarks involving semantic similarity tasks, such as STS (Semantic Textual Similarity) benchmarks [106].	Performs well in tasks involving large-scale data where efficiency outweighs the need for deep semantic understanding [104].

TABLE 3.1 – Comparison between Word Mover's Distance and Cosine Similarity.

3.3.4 Application of Word Embeddings with Word Mover's Distance (WMD)

Word embeddings, such as Word2Vec, have revolutionized natural language processing by representing words as dense vectors in continuous vector spaces . When combined with Word Mover's Distance (WMD), these embeddings offer powerful

tools for various text-based applications [107].

Semantic Text Similarity: By leveraging Word2Vec embeddings and WMD, applications can accurately measure the semantic similarity between text documents. This capability is particularly useful in tasks such as duplicate detection, plagiarism detection, and information retrieval [108].

Document Clustering: Word embeddings combined with WMD enable more sophisticated document clustering techniques. Instead of relying solely on lexical or syntactic features, clustering algorithms can now consider semantic relationships between documents, leading to more coherent clusters [109].

Text Classification: In text classification tasks, Word2Vec embeddings with WMD can improve the accuracy of classification models by capturing subtle semantic differences between classes. This is especially beneficial in domains where traditional bag-of-words approaches may struggle, such as sentiment analysis and topic modeling [110].

Cross-Lingual Text Alignment: Word embeddings and WMD facilitate cross-lingual text alignment by capturing semantic similarities between documents in different languages. This is crucial for tasks such as machine translation, cross-lingual information retrieval, and multilingual document summarization [111].

Named Entity Recognition and Entity Linking: Incorporating Word2Vec embeddings with WMD enhances named entity recognition (NER) and entity linking tasks by considering not only surface-level features but also semantic similarities between entities. This leads to more accurate identification and disambiguation of named entities in text [112].

3.3.4.1 Layman's Explanation of Word2Vec and Word Mover's Distance

This section introduces two important concepts used in our method: Word2Vec and WordMover's Distance (WMD). Imagine a world where words are points in a vast space. Words with similar meanings are clustered close together. This is what Word2Vec does - it creates a map for words based on how often they appear together in text. Each word gets its own unique location in this space.

We can then treat documents as collections of these word-locations. Now comes WMD, which helps us measure the distance between documents. It considers how much "effort" it takes to move the words in one document to match the words in another. Documents with similar meanings will have a smaller WMD, as their word-locations are naturally closer. Here's an analogy: Imagine documents as grocery lists. Word2Vec assigns each ingredient (word) a position in a giant supermarket (embedding space). WMD then tells us how much walking we need to do (distance) to get all the ingredients on one list (document) to match the other.

This approach allows us to leverage pre-trained word embeddings, which are like pre-made shopping lists for words. They capture the relationships between words and save us time (training) when working with documents [113].

Figure 3.2 shows an illustrative example on how to use WMD to capture the distance between two documents.

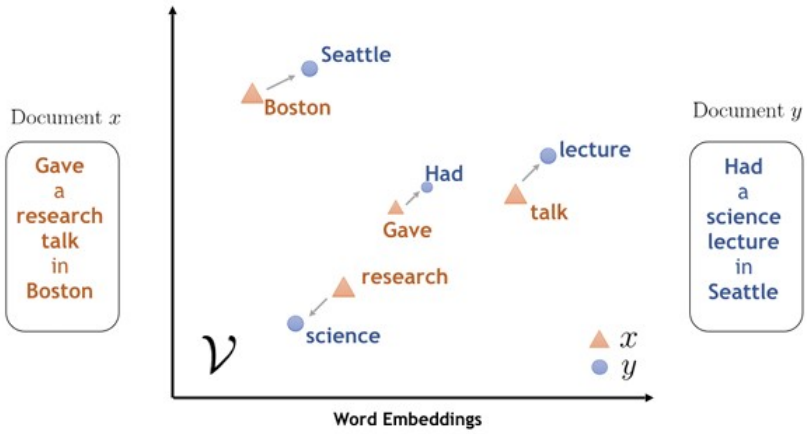

FIGURE 3.2 – An illustration of the WMD measures the distance between two documents [113].

3.3.4.2 Capturing Word Similarities and Relationships with Word2Vec

Examples made with the model consist of Google News texts, the words that are similar to each other are represented by close vectors. For example, dog and cat are represented in the areas close to each other. In addition, relations between words present close associations also in similar word phrases. Therefore, the relationship between 'fast', 'faster' and 'fastest' is similar to 'long', 'longer' and 'longest'. Using the model created with Word2vec, the similarities of the words can be reached. In the model created with Google news, the closest word to 'Man' is 'Woman' (with a similarity value of 0.69). In a certain group of words, it can distinguish the irrelevant word with doesnt_match function. The command doesnt_match ('blue red green yellow book') returns the word 'book' in response [114].

3.3.5 Utilizing Transformer Models: The Case of FastText

FastText, unlike many other language models, relies on a unique architectural paradigm. Here, we outline a three-stage process for enhancing FastText models: pre-training, post-training, and fine-tuning.

3.3.5.1 Pre-training

The initial training procedure of our proposed FastText enhancement follows the standard pre-training approach of FastText. FastText is known for its efficiency and effectiveness in text classification tasks, leveraging subword information to handle rare and misspelled words. In this stage, FastText is trained on a large general corpus to create a robust initial model [115].

3.3.5.2 Post-training

Instead of directly fine-tuning the pre-trained FastText model, we introduce a post-training stage. This stage involves further training the pre-trained FastText model on a domain-specific or task-specific corpus. This additional training helps the model adapt better to the nuances and biases of the specific domain, thus enhancing its performance on related tasks. The post-training stage updates the model

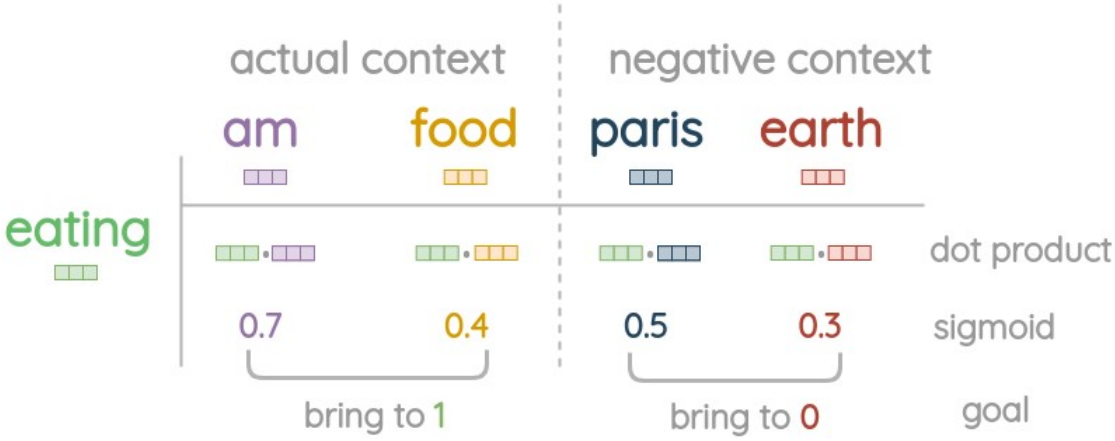

FIGURE 3.3 – A Visual Guide to FastText Word Embeddings [118].

parameters to better align with the intermediate task before moving on to fine-tuning for the target task [116].

3.3.5.3 Fine-tuning

In the fine-tuning stage, we initialize our enhanced FastText model with the post-trained parameters. This model is then further fine-tuned using a supervised dataset specific to the downstream task. Each downstream task thus benefits from a uniquely fine-tuned model that leverages the domain-specific enhancements made during the post-training stage. This process ensures that the model is not only pre-trained on a large general corpus but also specialized through post-training, and finally, optimized for specific tasks through fine-tuning [117].

3.4 Evaluation Metrics

Performance metrics such as precision, accuracy, recall, and F1-score are commonly used to evaluate the effectiveness of similarity evaluation and outlier detection methods. These metrics are derived from the contingency matrix (also known as the confusion matrix), which tabulates the true positive (TP), false positive (FP), true negative (TN), and false negative (FN) predictions [99].

Precision: Precision measures the proportion of true positive predictions among all positive predictions (TP / (TP + FP)).

Accuracy: Accuracy measures the proportion of correctly classified instances among all instances ((TP + TN) / (TP + FP + TN + FN)).

Recall (Sensitivity): Recall measures the proportion of true positive predictions among all actual positive instances (TP / (TP + FN)).

F1-score: F1-score is the harmonic mean of precision and recall, providing a balanced measure of model performance ((2 * Precision * Recall) / (Precision + Recall)).

Employing performance metrics based on the contingency matrix, researchers and practitioners can gain insights into the strengths and weaknesses of similarity

evaluation and outlier detection algorithms, facilitating informed decision-making and algorithm refinement [119].

3.5 Case Studies and Examples

This section presents real-world case studies and examples that highlight the practical application of similarity evaluation and outlier detection methods across various domains. Each case study provides insights into how these concepts are applied in different contexts and the impact they have on decision-making processes.

3.5.1 Related Work

3.5.1.1 Word2Vec: Distributed Representations of Words

1) Efficient Estimation of Word Representations in Vector Space (2013): In 2013, Tomas Mikolov and colleagues introduced Word2Vec, a groundbreaking method for learning vector representations of words. This innovation transformed natural language processing (NLP) by enabling a deeper semantic understanding of text. Word2Vec comprises two models: Continuous Bag of Words (CBOW) and Skip-gram. CBOW predicts a target word from its surrounding context words, while Skip-gram predicts context words from a target word. This approach captures semantic relationships between words, allowing tasks like word analogy and similarity to be performed with high accuracy [120].

2) Distributed Representations of Words and Phrases and Their Compositionality (2013): Building on their earlier work, Mikolov et al. extended Word2Vec to handle phrases and enhance the quality of vector representations. This improvement enables the model to understand multi-word expressions and idiomatic phrases, further refining its ability to capture the nuances of human language [120].

3.5.1.2 Word Mover's Distance: Measuring Document Similarity

From Word Embeddings to Document Distances (2015): In 2015, Matt Kusner and colleagues introduced the Word Mover's Distance (WMD), a novel approach to measuring the semantic distance between documents using word embeddings. WMD leverages Word2Vec representations to quantify how much one document needs to "travel" to reach another, in terms of word embeddings. This method excels in capturing semantic similarities between texts, outperforming traditional distance metrics in many applications, such as document classification and retrieval.[121]

3.5.1.3 FastText: Efficient Text Classification with Subword Information

Bag of Tricks for Efficient Text Classification (2016): FastText, introduced by Joulin et al. in 2016, revolutionized text classification with its efficient use of subword information. Unlike traditional word embedding methods, FastText represents words as bags of character n-grams, allowing it to handle out-of-vocabulary words and morphological variations effectively. FastText's ability to capture subword information contributes to its success in tasks such as sentiment analysis, topic classification, and language identification.[122]

Enriching Word Vectors with Subword Information (2017): In their 2017 paper, Bojanowski et al. delve deeper into the mechanisms behind FastText's effectiveness

by enriching word vectors with subword information. They propose a method to incorporate character-level information into word embeddings, enhancing the model's ability to capture semantic and morphological similarities between words. This enhancement further solidifies FastText as a powerful tool for text classification and related tasks.[123]

3.6 Conceptual Workflow for Text Similarity Analysis

We first start with a presentation of the workflow shown in Figure 3.4.

3.6.1 Data Loading and Preprocessing

a) Dataset Preparation: The process begins with preparing the dataset, which consists of a collection of text documents. If the dataset is stored in a JSON file, the data can be loaded from this file. Each line in the JSON file should contain an object with a 'text' field that holds the document's content. This step ensures that all necessary data is available for further processing.

b) Preprocessing Text: Once the dataset is loaded, the next step involves preprocessing the text data to ensure it is clean and consistent for analysis. The preprocessing function handles several tasks: converting all text to lowercase to maintain uniformity, tokenizing the text into individual words or tokens, removing common stop words (such as "the", "a", "an"), eliminating punctuation and digits, and lemmatizing the tokens to reduce words to their base forms (e.g., "running" to "run"). These steps help in reducing noise and focusing on the meaningful content of the text.

After defining the preprocessing steps, they are applied to each document in the dataset. This results in a collection of preprocessed documents where each document is represented by a list of cleaned and standardized tokens. This preprocessed dataset serves as the input for training the text similarity models.

3.6.2 Word2Vec and FastText Model Training

a) Training Word2Vec Model: Using the preprocessed dataset, a Word2Vec model is trained to generate word embeddings. This model captures the semantic relationships between words based on their context in the text. The training involves setting parameters such as the vector size (dimensionality of the word embeddings), the window size (the context window for learning word relationships), the minimum word count (filtering out infrequent words), and the number of worker threads for parallel processing. The trained Word2Vec model provides a vector representation for each word in the vocabulary.

b) Training FastText Model: Similarly, a FastText model is trained using the same preprocessed dataset. FastText extends the capabilities of Word2Vec by considering subword information, which helps in handling rare and out-of-vocabulary words more effectively. The training process for FastText involves similar parameters to those used for Word2Vec. The resulting FastText model also generates word embeddings that capture semantic relationships but with enhanced handling of word morphology.

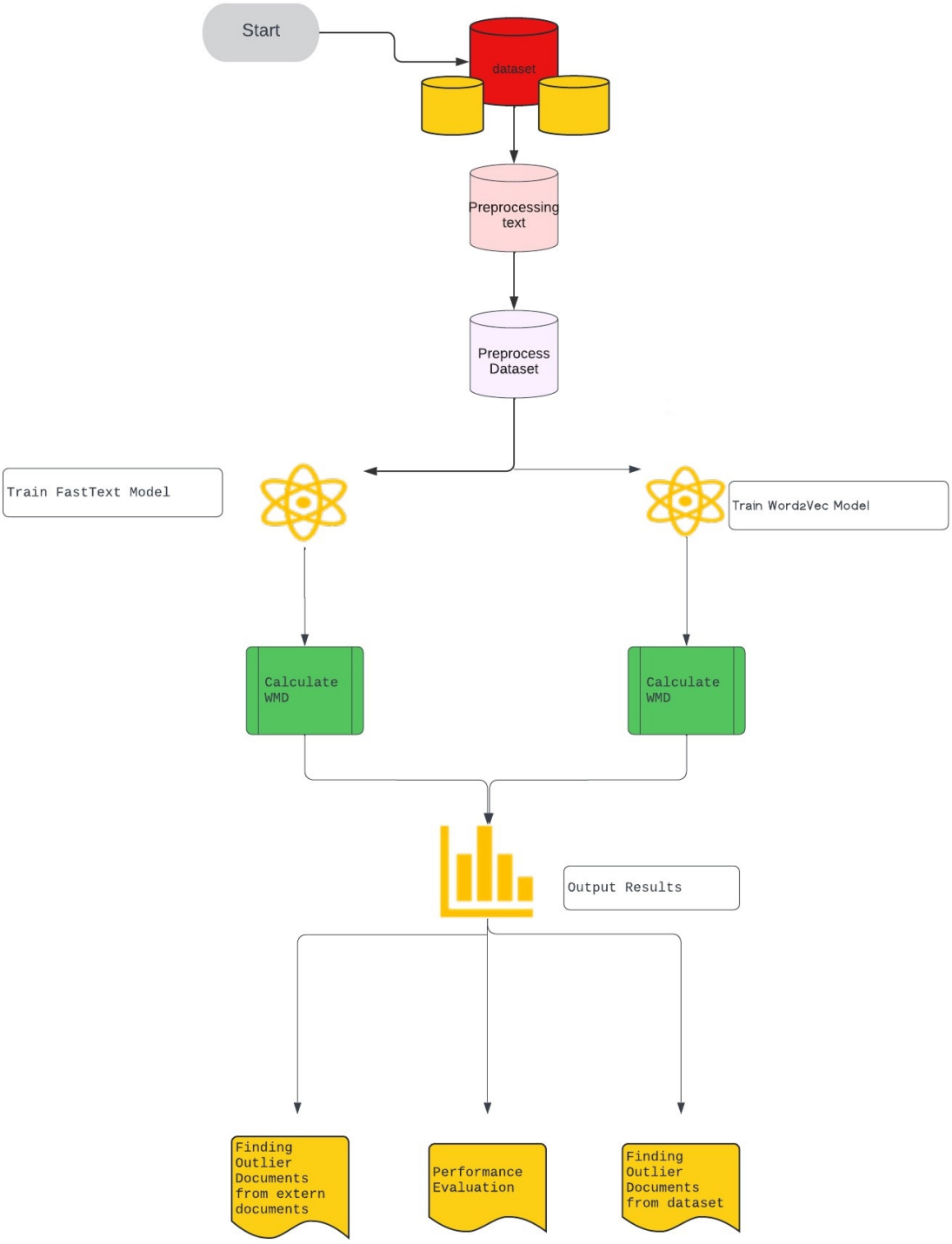

FIGURE 3.4 – Word2Vec/FastText Workflow using WMD to Detect Outliers.

3.6.3 Word Mover's Distance (WMD) Calculation

Calculating WMD: With both Word2Vec and FastText models trained, the next step is to calculate the Word Mover's Distance (WMD) between an input sentence and each document in the dataset. The input sentence is first preprocessed using the same steps as the dataset. For each document, the WMD is computed using the word embeddings from the respective models. WMD measures the minimum distance required to transform the word distribution of the input sentence to that of the document. This calculation is performed separately for both the Word2Vec and FastText models. The minimum WMD value for each model is tracked to identify the closest match between the sentence and the documents.

3.6.4 Sentence Similarity Evaluation

Based on the calculated WMD values, the similarity between the input sentence and each document is evaluated. Predefined thresholds are used to categorize the level of similarity: if the WMD is below a low threshold, the documents are considered "Highly Similar"; if the WMD is between the low and high thresholds, the documents are "Somewhat Similar"; and if the WMD exceeds the high threshold, the documents are "Not Similar". This categorization helps in understanding the degree of similarity based on the distance metrics provided by the models.

3.6.5 Performance Evaluation

To assess the effectiveness of the similarity evaluation, performance metrics such as accuracy, precision, recall, and F1-score are calculated. These metrics are based on a set of labeled sentences that serve as ground truth. The performance evaluation provides insights into how well the models and similarity thresholds perform in categorizing text similarity, allowing for potential adjustments and improvements.

3.6.6 Finding Outlier Documents

Finally, the process includes identifying outlier documents that are significantly different from the input sentence. For each model (Word2Vec and FastText), documents with WMD values above a high threshold are considered outliers. The process involves calculating the WMD for each document and the input sentence, and tracking the closest outlier document (the one with the minimum WMD above the threshold). This step helps in detecting documents that deviate notably from the norm, providing valuable insights into anomalies in the dataset.

3.7 Conclusion

This chapter delved into the fundamental concepts of similarity evaluation and outlier detection. We explored various techniques for measuring similarity and identifying data points that deviate significantly from the norm. Through compelling case studies across diverse domains, we witnessed the practical applications of these methods in recommender systems, fraud detection, anomaly detection, customer segmentation, and image retrieval.

Our proposed method leverages word embeddings and Word Mover's Distance (WMD) to overcome the limitations of traditional similarity measures. Unlike cosine similarity, which only considers the angle between vectors, WMD accounts for the

semantic distance between words, providing a richer and more accurate measure of text similarity. Word2Vec and FastText enhance this approach by generating word embeddings that capture the contextual meaning of words, improving the robustness of similarity evaluation.

Chapter 4

Implementation

4.1 Introduction

In this chapter, we're diving into the practical side of our text similarity and outlier detection system. We'll translate our theoretical framework and methodologies into action, bridging the gap between concept and real-world application.

We'll start by tackling the preprocessing tasks needed to prepare our textual data for analysis. This involves steps like reading data from JSON files, tokenization, removing stop words and punctuation, and lemmatizing words to ensure consistency and quality.

Next, we'll explore the selection and training of machine learning models. We'll focus on Word2Vec for text similarity, training it with our preprocessed dataset. Additionally, we'll incorporate FastText embeddings to deepen our understanding of text similarity, evaluating each model's effectiveness in capturing semantic meaning.

Once the models are trained, we'll delve into measuring text similarity and detecting outliers. We'll use the Word2Vec model to calculate the Word Mover's Distance (WMD) and explore how FastText embeddings combined with WMD enhance our analysis.

To wrap up, we'll implement a user-friendly interface using Tkinter. This interface will allow users to upload data, process it, and visualize results seamlessly. With detailed explanations and code snippets, this chapter serves as a comprehensive guide to the implementation process, addressing challenges and presenting solutions. By the end, readers will have the knowledge to replicate or extend our system with confidence.

4.2 System Architecture

4.2.1 Development Environment

4.2.1.1 Hardware Configuration

— Processor: AMD Ryzen 5 5600G with Vega 7 Graphics

— Memory: 16GB DDR4 RAM

— Storage: 500GB NVMe SSD

4.2.1.2 Software Framework

In order to develop our system, we used a set of software tools and frameworks that facilitate efficient coding, debugging, and execution of our machine learning models.

```
import json
import tkinter as tk
from tkinter import filedialog, messagebox, Text, Scrollbar, RIGHT, Y, BOTH, Frame
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import string
from gensim.models import Word2Vec
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
from collections import defaultdict
```

FIGURE 4.1 – Essential Libraries.

a) Operating System: The implementation was carried out on a Windows 10 operating system. However, the described setup and code are compatible with other operating systems, such as Linux and macOS, with minimal adjustments.

b) Python Programming Language: It is an open source program language, easy and convenient to use. It is important that nothing needs to be compiled for the function. Python allows the programmers to focus on the font that appears on the screen that does not use the font. so, these programs are available at the lowest temperatures in a foreign language.

c) Anaconda Distribution: Anaconda was used as the primary package manager and environment management tool. It simplifies the installation of necessary libraries and ensures that dependencies are properly managed.

4.2.1.3 Integrated Development Environment (IDE)

a) Visual Studio Code (VS Code): VS Code was used as the main IDE for writing and debugging code. Its extensions for Python, such as the Python extension by Microsoft, provide powerful features for coding, testing, and debugging.

b) Python Libraries and Frameworks: Several libraries (Figure 4.1) were utilized to facilitate different aspects of the implementation:

- Natural Language Toolkit (NLTK): For text pre-processing tasks such as tokenization, stop words removal, and lemmatization.

- Gensim: For training and utilizing the Word2Vec model.

- Scikit-learn: For calculating evaluation metrics such as confusion matrix, accuracy, precision, recall, and F1 score.

- Tkinter: For creating the graphical user interface.

c) Data Storage : JSON files were used to store and manage textual data. This format is lightweight and easy to parse in Python, making it suitable for our needs.

4.2.2 System Workflow

The system follows a structured workflow, starting from data loading to model training and evaluation, and finally presenting the results through a user interface.

a) Data Loading: Textual data is loaded from JSON files. Each document is extracted and stored for further processing.

b) Data Pre-processing: The loaded data undergoes several pre-processing steps to ensure quality and consistency: Tokenization, Stop words removal, Punctuation removal, Lemmatization.

c) Model Training:

1. **Word2Vec Model:** Trained on the preprocessed dataset to generate word embeddings.
2. **FastText Model**: Utilized for generating deep contextual embeddings of sentences.
3. **Similarity Measurement and Outlier Detection:** Word Mover's Distance (WMD) calculated using Word2Vec embeddings to measure text similarity.

d) User Interface: Developed using Tkinter, the interface allows users to: Upload data ,Process the data ,View similarity measurements ,Detect outliers ,Display evaluation metrics

4.2.3 Data Preparation

4.2.3.1 Data sources and collection methods

Since we are working on a new idea, suitable datasets for our work are almost unavailable. After extensive searching, we were able to find the following:
yelp_academic_dataset_tip: This dataset is a subset of Yelp's data, originally compiled for the Yelp Dataset Challenge. This challenge provides students with an opportunity to conduct research or analysis on Yelp's data and share their findings.

Additionally, we created two datasets: one for film and another for computer science.

Each dataset consists of similar documents and some outlier documents, with each document labeled as either 0 (similar document) or 1 (outlier document). This labeling is essential for evaluation purposes.

For each dataset, we created numerous external documents with their respective labels. These external documents are used to test the datasets and incorporate them into the ratings calculation.

4.2.3.2 Data preprocessing steps

To prepare the textual data for further processing, several pre-processing steps are performed as shown in Figure 4.2.

a) Reading Data: The data is read from JSON files which contain the raw textual data.

```
def preprocess_text(self, text):
    stop_words = set(stopwords.words('english'))
    lemmatizer = WordNetLemmatizer()
    tokens = word_tokenize(text.lower())
    tokens = [lemmatizer.lemmatize(token) for token in tokens if token not in stop_words a
    return tokens

def preprocess_dataset(self):
    self.preprocessed_data = []
    for document in self.json_data:
        preprocessed_document = self.preprocess_text(document)
        self.preprocessed_data.append(preprocessed_document)
```

FIGURE 4.2 – Data preprocessing steps.

b) Tokenization: The text is converted to lowercase to maintain uniformity.

The text is tokenized into individual words. Tokenization involves splitting the text into smaller units called tokens, typically words.

c) Removing Stop-Words and Punctuation: Stop words (common words like 'the', 'is', etc., that do not contribute much to the meaning) are removed. Punctuation marks are also removed as they do not contribute to the text's semantic meaning.

d) Lemmatization: Words are lemmatized to convert them to their base or root form. This helps in reducing inflectional forms and derivationally related forms of a word to a common base form. For example, 'running' is lemmatized to 'run'.

Hereafter is the explanation of the preprocessing Python code:

- Tokenization: word_tokenize(text.lower()): Tokenization is the process of breaking down text into individual words or tokens. In the provided code, the text is first converted to lowercase using text.lower(), and then word_tokenize is applied to split the text into tokens. This step is crucial for further text processing tasks as it helps in analyzing the text at the word level.
- Lowercasing: text.lower(): Lowercasing converts all characters in the text to lowercase. This step ensures that the text is uniform, eliminating discrepancies between words that are capitalized and those that are not. For example, 'Dog' and 'dog' would be treated as the same token after lowercasing.
- Stopword Removal: token not in stop_words: Stopwords are common words (e.g., 'the', 'is', 'in') that usually do not contribute significantly to the meaning of a text. Removing stopwords helps reduce the dimensionality of the data and focus on more meaningful words. In the code, tokens that are in the set of predefined stopwords are excluded from further processing.
- Punctuation Removal: token not in string.punctuation: Punctuation marks do not carry significant meaning in most text analysis tasks. The code ensures that tokens which are punctuation marks are removed from the text. This helps in cleaning the text and focusing on the actual words.
- Lemmatization: lemmatizer.lemmatize(token): Lemmatization is the process of converting words to their base or root form. For instance, 'running' is converted to

```
def process_data(self):
    if not self.json_data:
        messagebox.showerror("Error", "Please load a JSON file first!")
        return
    self.preprocess_dataset()
    self.word2vec_model = Word2Vec(sentences=self.preprocessed_data, vector_size=100, window=5, min_count=1, workers=4)
    self.word2vec_model.save('word2vec_model.bin')
    messagebox.showinfo("Success", "Data processed and Word2Vec model trained successfully!")
```

FIGURE 4.3 – Training the Word2Vec model.

'run', and 'better' is converted to 'good'. This helps in reducing different forms of a word to a single representation, thereby simplifying the text data.
- Digit Removal: not token.isdigit(): Digit removal ensures that numeric characters are excluded from the text. This step is often necessary when numerical values do not contribute to the analysis or are irrelevant to the text processing task.
- Stopword Filtering: Handled as part of the initial filtering of tokens: As mentioned in step c, stopword filtering is integrated into the token filtering process. Tokens that match any of the stopwords are removed from the list of tokens during initial filtering.
- Storage of Preprocessed Data: self.preprocessed_data.append(preprocessed_document) after processing each document, the preprocessed data (i.e., the list of cleaned and tokenized words) is stored in self.preprocessed_data. This ensures that the processed text is saved for further analysis or model training.

4.3 Implementation of Text Similarity Using Word2Vec

In this project, Word2Vec is leveraged to measure the similarity between texts. By training a Word2Vec model on the preprocessed data, we can transform words into vectors and compute the similarity between different texts using metrics like the Word Mover's Distance (WMD). This approach enables accurate detection of similar texts and outliers within the dataset, making it a critical component of the Text Similarity and Outlier Detection application.

4.3.1 Training the Word2Vec Model

Once the data is preprocessed, the next step is to train the Word2Vec model. The model learns to generate vector representations of words based on their context within the training data. The trained model can then be used to compute text similarities. Figure 4.3 shows how to train and save the Word2Vec model.
a) Check for Data: Ensures that JSON dataset is loaded before proceeding.

b) Preprocess Dataset: Calls the preprocess_dataset method to clean the data.

c) Train Model: Initializes and trains the Word2Vec model using the preprocessed data.

d) Save Model: Saves the trained model to a file for future use.

```
def calculate_wmdistance(self, sentence):
    sentence_tokens = self.preprocess_text(sentence)
    if len(sentence_tokens) == 0:
        return float('inf')
    min_distance = float('inf')
    for document in self.preprocessed_data:
        distance = self.word2vec_model.wv.wmdistance(document, sentence_tokens)
        if distance < min_distance:
            min_distance = distance
    return min_distance
```

FIGURE 4.4 – Calculate Word Mover's Distance.

```
self.word2vec_model = Word2Vec.load('word2vec_model.bin')
```

FIGURE 4.5 – Saving Wor2Vec Model.

4.3.2 Calculate Word Mover's Distance

After training the model, the next step is to calculate the Word Mover's Distance (WMD) between a given sentence and preprocessed documents in the dataset (Figure 4.4). WMD measures semantic similarity by calculating the minimum cumulative cost to transform one set of word embeddings into another. The code preprocesses the input sentence through tokenization, lowercasing, and filtering out stop words, punctuation, and digits. If the sentence is empty after preprocessing, it returns infinity. The function then iterates through each preprocessed document, computes the WMD.

4.3.3 Saving and Loading the Trained Model

To ensure the trained Word2Vec model can be reused without retraining, it is saved to a file after training. The saved model can be loaded later for further analysis.

Figures 4.5 and 4.6 show how to load and save a trained model, respectively.

4.4 Implementation of Text Similarity Using FastText

After finishing the Word2Vec code , we start with another method "FastText", and we kept all the functionality. The only addition is the training of FastText.

The function shown in Figure 4.7 checks for JSON data, preprocesses the data, trains the FastText model, saves the model, and displays a success message.

4.5 Outlier Detection

In our project Outlier detection is crucial for several reasons . We mention some of them here:

```
self.word2vec_model.save('word2vec_model.bin')
```

FIGURE 4.6 – Loading word2vec.

```python
def process_data(self):
    if not self.json_data:
        messagebox.showerror("Error", "Please load a JSON file first!")
        return
    self.preprocess_dataset()
    self.model = FastText(sentences=self.preprocessed_data, vector_size=100, window=5, min_count=1, workers=4)
    self.model.save('fasttext_model.bin')
    messagebox.showinfo("Success", "Data processed and FastText model trained successfully!")
```

FIGURE 4.7 – Processing Data and Training FastText Model.

```python
def check_sentences(self):
    if not self.word2vec_model:
        messagebox.showerror("Error", "Please process the data first!")
        return

    self.text_area.delete('1.0', tk.END)
    for sentence in self.sentences:
        distance = self.calculate_wmdistance(sentence)
        similarity_label, similarity_percentage = self.evaluate_similarity(distance, 0.9, 1.2)
        self.text_area.insert(tk.END, f"Sentence: '{sentence}'\n")
        self.text_area.insert(tk.END, f"Distance: {distance:.2f}\n")
        self.text_area.insert(tk.END, f"Similarity: {similarity_label} ({similarity_percentage:.2f}%)\n")
        self.text_area.insert(tk.END, "-" * 30 + "\n")
```

FIGURE 4.8 – Outlier Detection from Sentences.

a) Data Quality: Outliers can indicate errors or anomalies in data collection, transcription, or entry. Detecting these outliers helps maintain the integrity and accuracy of the dataset.

b) Analysis Accuracy: Outliers can distort statistical analyses and machine learning models. Removing or properly handling outliers ensures more reliable and valid analytical outcomes.

c) Identifying rare events: in some cases, outliers may represent rare but significant events or phenomena that require further investigation.

d) Domain-Specific Insights: Outliers can reveal unexpected patterns or trends specific to your field of study. Analyzing these outliers can lead to new discoveries and a deeper understanding of the underlying processes at play in your data.

4.5.1 Outlier Detection from Sentences

After uploading the sentences and training the model and calculate the distance, we now start to check if the sentence is outlier or similar. These are the two thresholds used to categorize the similarity: Values less than 0.9 are considered "Very Similar", values between 0.9 and 1.2 are "SomhowSimilar", and anything above 1.2 is considered "Not Similar".

Figure 4.8 illustrates this operation.

Sometimes we encounter sentences that are somewhat similar—too similar to be classified as outliers but not similar enough to be clearly considered similar. When the distance falls between 0.9 and 1.2, it led us to create the following section.

```python
def refind_similarity(self):
    if not self.word2vec_model:
        messagebox.showerror("Error", "Please process the data first!")
        return

    self.text_area.delete('1.0', tk.END)
    for sentence in self.sentences:
        distance = self.calculate_wmdistance(sentence)
        similarity_label, similarity_percentage = self.evaluate_similarity(distance, 1, 1.2)

        if similarity_label == "Somewhat Similar":
            similar_distances = []
            for doc in self.preprocessed_data:
                similar_distances.append(self.word2vec_model.wv.wmdistance(self.preprocess_text(sentence), doc))
            if similar_distances:
                refined_distance = (distance +min( similar_distances))/  2
                similarity_label, similarity_percentage = self.evaluate_similarity(refined_distance, 1, 1.001)

        self.text_area.insert(tk.END, f"Sentence: '{sentence}'\n")
        self.text_area.insert(tk.END, f"Distance: {distance:.2f}\n")
        self.text_area.insert(tk.END, f"Refined Distance: {refined_distance:.2f}\n" if similarity_label == "Somewhat Similar" else "")
        self.text_area.insert(tk.END, f"Similarity: {similarity_label} ({similarity_percentage:.2f}%)\n")
        self.text_area.insert(tk.END, "-" * 30 + "\n")
```

FIGURE 4.9 – Refind Similarity.

4.5.2 Refining Sentence Similarity Analysis for "Somewhat Similar" Cases

For we can say to that kind of documents if it's similar or outlier, we need to calculate the distance between him and the similar sentences and combine it with the last distance and we divide them by 2, as shin in Equation 4.1.

$$D_{\text{combined}} = \frac{D_{\text{initial}} + D_{\text{refined}}}{2} \tag{4.1}$$

a) Initial Distance ($D_{initial}$): This is the initial measure of how far the sentence is from the documents in the training set. It is calculated using the Word Mover's Distance, which measures the dissimilarity between two sets of words.

b) Refined Distance ($D_{redefined}$): To get a more precise measure of similarity, we compute the distance between the "Somewhat Similar" sentence and the most similar sentence within the training set. This minimum distance is the closest match in terms of similarity.

c) Combined Distance ($D_{combined}$): By averaging the initial distance and the refined minimum distance, we obtain a more accurate representation of similarity. This combined distance helps in making a more informed decision about whether the sentence is truly similar or an outlier.

By refining the sentence similarity analysis, we enhance the accuracy, sensitivity, and robustness of our text classification, ensuring more reliable and meaningful results. This process helps us make more informed decisions about the similarity and outlier status of sentences, ultimately improving the quality of our text analysis , as shown in Figure 4.9.

4.5.3 Identify Anomalies in Dataset

We have also included some outliers in the datasets and this is for detect them, as shown in Figure 4.10. This method ensures that the identified outliers are those that are genuinely different from the majority of the dataset, which helps maintain the integrity and quality of the data for further analysis or processing.

```
def find_outliers(self):
    if not self.word2vec_model:
        messagebox.showerror("Error", "Please process the data first!")
        return

    high_threshold = 1.25
    outlier_documents = defaultdict(float)

    for i, doc1 in enumerate(self.preprocessed_data):
        distances = []
        for j, doc2 in enumerate(self.preprocessed_data):
            if i != j:
                distance = self.word2vec_model.wv.wmdistance(doc1, doc2)
                distances.append(distance)

        avg_distance = sum(distances) / len(distances) if distances else float('inf')

        if avg_distance > high_threshold:
            outlier_documents[' '.join(doc1)] = avg_distance

    self.text_area.delete('1.0', tk.END)
    if outlier_documents:
        self.text_area.insert(tk.END, "Outlier Documents from the dataset:\n")
        for doc, wmd_score in outlier_documents.items():
            self.text_area.insert(tk.END, f"Document: {doc}\n")
            self.text_area.insert(tk.END, f"Average Word Mover's Distance: {wmd_score:.2f}\n")
            self.text_area.insert(tk.END, "-" * 30 + "\n")
    else:
        self.text_area.insert(tk.END, "No outlier document found.\n")

def load_sentences_and_labels(self):
```

FIGURE 4.10 – Find outlier from dataset.

4.6 User Interface (UI) Development

The tkinter package ("Tk interface") is the standard Python interface to the Tcl/Tk GUI toolkit. Both Tk and tkinter are available on most Unix platforms, including macOS, as well as on Windows systems [122].

The interface consists of a set of buttons and a space showing results, as shown in Figure 4.11.

a) Load JSON File Button: This button allows users to load a JSON file containing textual data. Upon clicking, it opens a file dialog where users can select the JSON file from their system. The loaded data is then parsed and stored for further processing.

b) Process Data Button: This button triggers the preprocessing of the loaded textual data. It includes steps like tokenization, stopword removal, and lemmatization. Additionally, it generates embeddings for the preprocessed text using either Word2Vec or BERT models.

c) Check Similarity Button: This button allows users to check the similarity of input sentences against the preprocessed dataset. Users can input sentences, and the system will evaluate and display their similarity scores relative to the dataset.

d) Find Outliers Button: This button identifies and displays outlier documents in the dataset using the model. Outliers are documents that significantly differ from the rest of the dataset based on their embeddings.

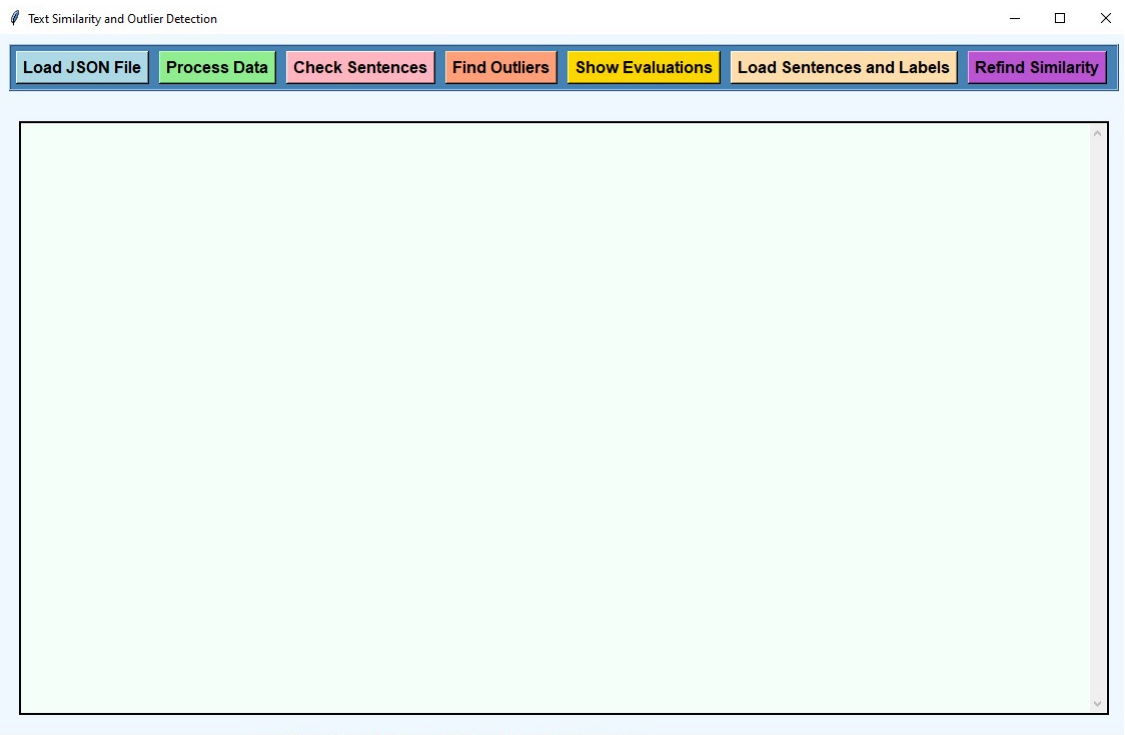

FIGURE 4.11 – Main User Interface of the Developed Application.

e) Load Sentences and Labels Button: This button allows users to load a file containing sentences and their corresponding labels. These files are used for evaluating the similarity detection accuracy.

f) Show Evaluations Button:
This button evaluates the similarity detection system's performance using loaded sentences and labels. It calculates and displays various evaluation metrics such as confusion matrix, accuracy, precision, recall, and F1 score.

4.6.1 Results Examples

a) Detecting Outlier From Sentences:
As shown in Figure 4.12, for each sentence, the application provides:

- Sentence: The actual text of the sentence being evaluated.

- Distance: The calculated distance (Word Mover's Distance, WMD) between the sentence and the dataset.

- Similarity: A label indicating how similar the sentence is to others, along with a percentage score.

b) Correcting the Documents of Somehow Similar:
The highlighted sentence about robotics shown in Figure 4.13, is identified as somewhat similar but not highly related to the dataset. After applying the formula shown in Equation 4.1, Refining Sentence Similarity Analysis for "Somewhat Similar Cases" we can now know that the SomwhatSimilar sentence is similar or outlier.

a) Find Outlier From Dataset:

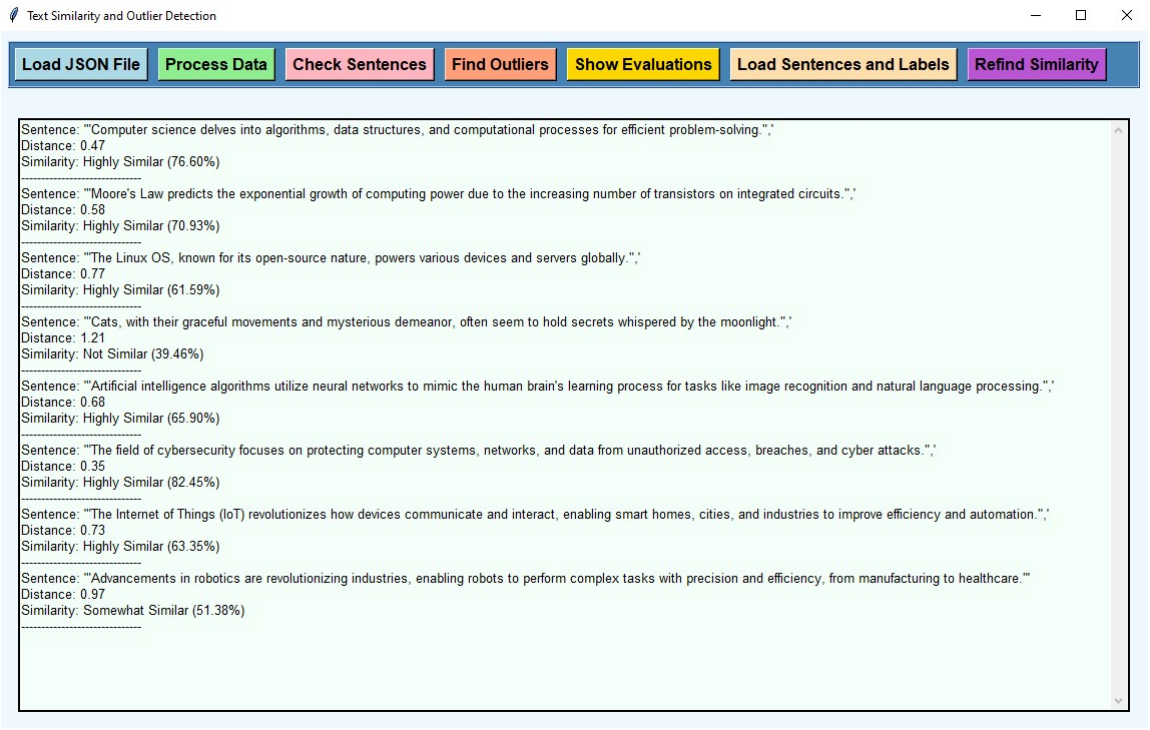

FIGURE 4.12 – Detecting Outliers among Sentences.

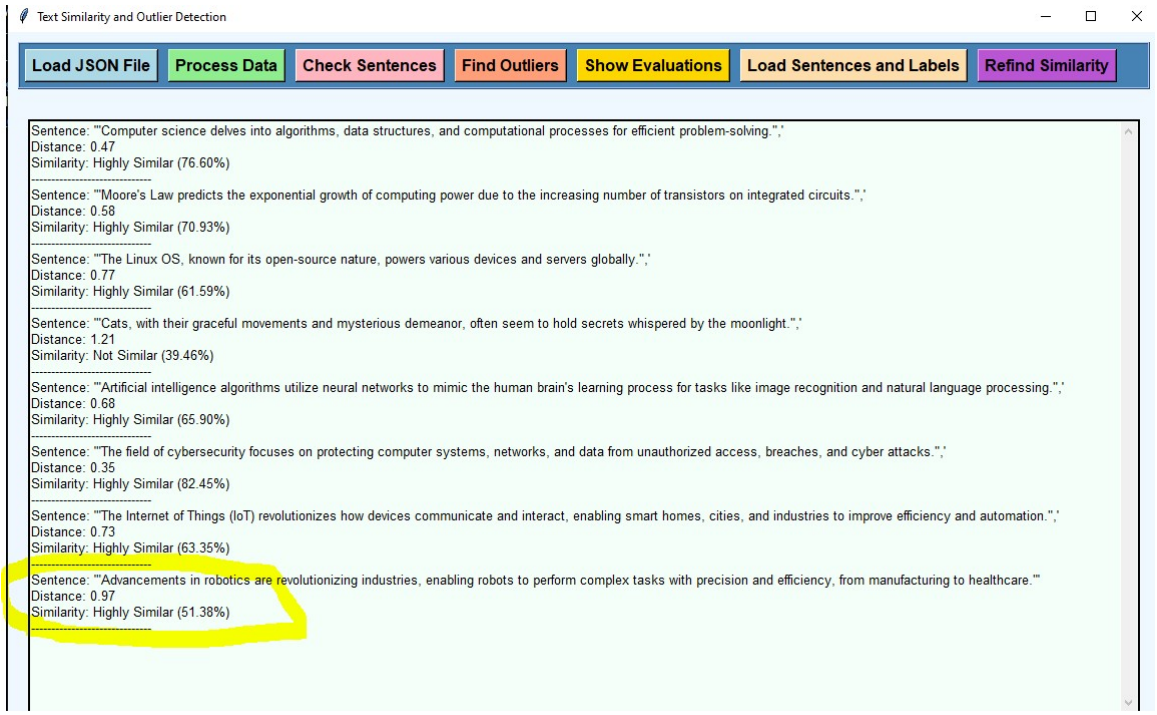

FIGURE 4.13 – Correcting the Classification of Documents with Somewhat Similarity.

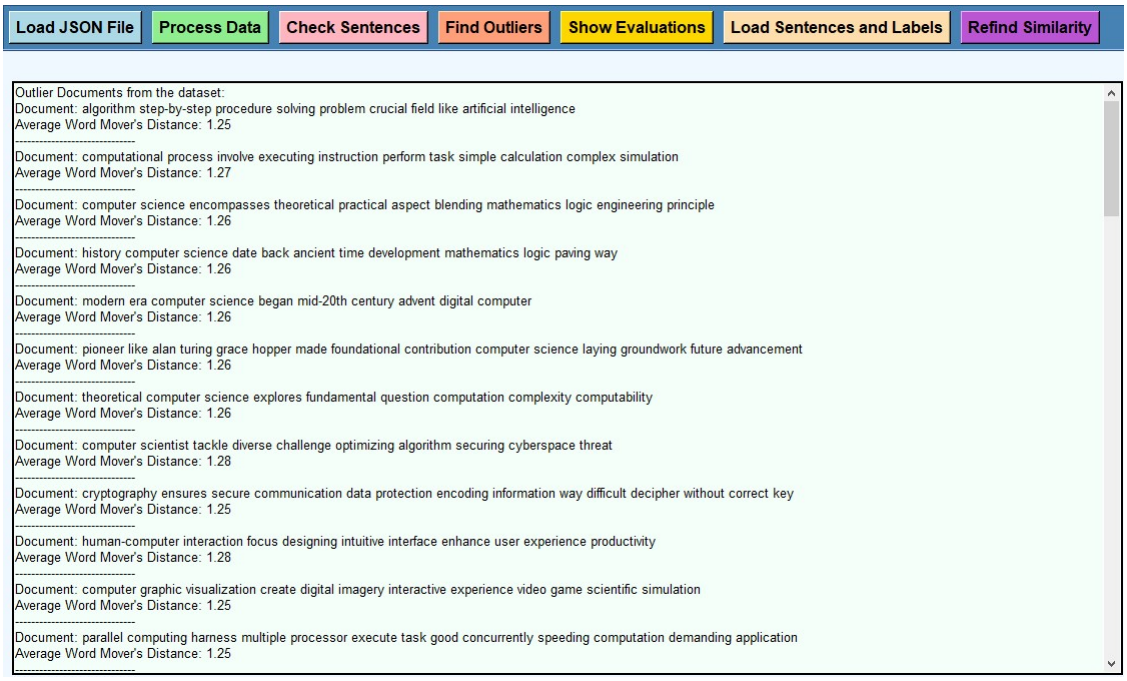

FIGURE 4.14 – Identifying Outliers in the Dataset.

As shown in Figure 4.14, this button runs the algorithm to identify outliers based on predefined criteria. In this context, outliers are documents that have a significantly different semantic content compared to the rest of the dataset, as measured by the Word Mover's Distance (WMD).

4.7 Evaluation Metrics

Discussion on how these metrics help in assessing the performance of the models:

a) Accuracy gives a general overview of the model's performance but can be misleading in cases of imbalanced datasets where one class might dominate. It is best used when the cost of false positives and false negatives is roughly the same.

b) Precision is critical when the cost of false positives is high. For instance, in a spam detection system, precision would tell us how many of the messages flagged as spam are actually spam.

c) Recall is important when the cost of false negatives is high. For example, in a disease detection system, recall would tell us how many of the actual disease cases were correctly identified.

d) F1 Score provides a balance between precision and recall, especially useful when there is an uneven class distribution or when both false positives and false negatives carry significant costs.

Example: Figure 4.15 shows how these metrics indicate how well the model can identify outlier documents versus normal ones and classify different types of text data

FIGURE 4.15 – Evaluation Metrics

accurately. The confusion matrix helps diagnose which classes are being confused the most, guiding further improvements in the model.

4.8 Conclusion

In this chapter, we have provided a comprehensive overview of the implementation process for our text similarity and outlier detection system. We began by discussing the system architecture, detailing the development environment, software frameworks, and integrated development environment (IDE) used.

We then delved into the data preparation phase, where we outlined data sources, collection methods, and the preprocessing steps necessary to clean and homogenize the textual data. This included tokenization, stop-word removal, punctuation removal, and lemmatization.

The implementation of text similarity using Word2Vec and FastText models was thoroughly covered, including model training, calculation of Word Mover's Distance (WMD), and the process of saving and loading trained models. We also explored outlier detection methods, such as identifying outliers from sentences, refining similarity analysis for "somewhat similar" cases, and detecting anomalies in the dataset.

Furthermore, we discussed the development of the user interface (UI) using Tkinter, which provides users with an intuitive and interactive experience. Results examples were provided to demonstrate how the UI displays similarity scores, identifies outliers, and evaluates model performance using various metrics.

General Conclusion

The research presented in this project highlights the efficacy of advanced word embedding techniques, specifically Word2Vec and the Word Mover's Distance (WMD), in detecting outliers within textual datasets. Through the generation of dense vector representations and the application of WMD, the project successfully identifies documents that exhibit significant deviations from the norm, showcasing their potential as outliers.

The experiments conducted on benchmark datasets validate the approach, demonstrating its capability to accurately and efficiently detect anomalies. This methodology's strength lies in its ability to handle semantic richness and provide a detailed measure of document similarity. As such, it offers a valuable tool for various applications, including fraud detection, cybersecurity, and healthcare, where identifying anomalies is crucial.

Future work may explore integrating additional machine learning techniques to further enhance outlier detection capabilities, ensuring more comprehensive and accurate results.

Bibliography

Chapter 1:

[1]Spotfire. (n.d.). What is outlier detection. Retrieved from https://www.spotfire.com/glossary/what-is-outlier-detection.

[2]Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2016). Bag of Tricks for Efficient Text Classification. arXiv preprint arXiv:1603.05201.

[3]DeepAI. (n.d.). Outlier Detection. Retrieved from https://deepai.org/machine-learning-glossary-and-terms/outlier-detection.

[4]Gama, J., and Pedroso, A. M. (2014). Survey on conceptual learning methods for outlier detection. Knowledge and Information Systems, 39(2), 313-344.

[5]Song, X., Wu, M., Zhu, C., Wang, H., and Li, W. (2007, June). Conditional anomaly detection. In Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 937-945). ACM.

[6]Chandola, V., Banerjee, A., and Kumar, V. (2007). Anomaly detection: A survey. ACM Computing Surveys (CSUR), 41(3), 1-58.

[7]Grubbs, R. K. (2011). Practitioners' guide to outlier detection. Springer Science and Business Media.

[8]Liu, L., Ting, K. M., and Zhou, Z. H. (2012). Isolation-based anomaly detection. ACM Transactions on Knowledge Discovery from Data (TKDD), 6(1), 1-39.

[9]Aggarwal, C. C., and Charu, C. (2015). Data clustering and outlier detection. Springer.

[10]Hawkins, D. M. (1980). Identification of outliers. Chapman and Hall.

[11]Usherbrooke University. (n.d.). Score-Z. Retrieved from https://psychometrie.espaceweb.usherbrook

[12]Liu, F. T., and Ting, K. M. (2009). Isolation Forest. In Proceedings of the 8th IEEE International Conference on Data Mining.

[13]Liu, L., Ting, K. M., and Zhou, Z. H. (2012). Isolation-based anomaly detection. ACM Transactions on Knowledge Discovery from Data (TKDD), 6(1), 1-39.

[14]Liu, F. T., Ting, K. M., and Zhou, Z. H. (2008). Isolation Forest. In Proceedings of the 8th IEEE International Conference on Data Mining.

[15]Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., and Williamson, R. C. (2001). Learning with kernels: Support vector machines, regularization, optimization, and beyond. MIT Press.

[16]Bolton, D., and Hand, D. (2015). Statistical fraud detection: A review. Statistical Science, 30(2), 235-255.

[17]Lazarevic, A., Ertoz, L., Kumar, V., Srivastava, A., and Kumar, P. (2005). A comparative study of anomaly detection schemes in network intrusion detection. SIAM Journal on Data Mining, 1(2), 213-232.

[18]Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. ACM Computing Surveys (CSUR), 41(3), 1-58.

[19]National Library of Medicine. (2024, May 27). Detection and explanation of anomalies in healthcare data.

[20]Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. ACM Computing Surveys, 41(3), 1-58.

[21]Chandola, V., Lazarevic, A., and Kumar, V. (2009). Anomaly detection: A survey. ACM Computing Surveys (CSUR), 41(3), 15.

[22]Hodge, V. J., and Austin, J. (1993). A survey of outlier detection methods. Pattern Recognition, 26(9), 955-975.

[23]Pang, G., et al. (2019). Deep learning for anomaly detection: An overview. arXiv preprint arXiv:1905.11303.

[24]Zhou, J., et al. (2019). Anomaly detection with LSTM based encoder-decoder. arXiv preprint arXiv:1901.09921.

[25]Chandola, V., Lazarevic, A., and Kumar, V. (2004). Feature selection for anomaly detection. ACM SIGKDD Explorations Newsletter, 6(1), 29-39.

[26]Bengio, Y. (2014). Learning deep representations: A recurrent neural network perspective. Communications of the ACM, 57(8), 67-78.

[27]Bolton, D., and Hand, D. (2015). Statistical fraud detection: A review. Statistical Science, 30(2), 235-255.

[28]Yu, H., et al. (2018). Deep learning for anomaly detection with scarce labeled data. In Proceedings of the IEEE International Conference on Data Mining (ICDM). IEEE.

[29]Rawat, W., and Singh, D. (2017). Deep convolutional neural networks for image classification: A comprehensive review. Neural Computation, 29(9), 2352-2443.

[30]Samek, W., et al. (2019). Explainable artificial intelligence (XAI) in healthcare: The balance between the explainable and the actionable. Health Informatics Journal, 25(1), 209-219.

[31]Chandola, V., Lazarevic, A., and Kumar, V. (2004). Feature selection for anomaly detection. ACM SIGKDD Explorations Newsletter, 6(1), 29-39.

[32]IBM. (n.d.). What is Explainable AI (XAI)? Retrieved from https://research.ibm.com/topics/expla

[33]DARPA. (n.d.). Explainable Artificial Intelligence (XAI). Retrieved from https://www.darpa.mil/program/explainable-artificial-intelligence.

[34]Trittinbach, H., et al. (2018). An Overview and a Benchmark of Active Learning for Outlier Detection with One-Class Classifiers. arXiv preprint arXiv:2207.05286.

[35]Rajpurkar, P., et al. (2016). SQuAD: 1.0 question answering dataset. arXiv preprint arXiv:1606.05250.

[36]Abe, N. (n.d.). Outlier Detection by Active Learning. Retrieved from https://arxiv.org/pdf/1601.0

[37]Zolanvari, M., et al. (2018). Context-Aware Outlier Detection in Streaming Data. ScienceDirect. Retrieved from https://www.sciencedirect.com/science/article/pii/S0888613X23002

[38]Liang, W., et al. (2019). Federated Outlier Detection. arXiv preprint arXiv:2209.04184. Retrieved from https://arxiv.org/pdf/2209.04184.

Chapter 2:

[39] Data Analytics Post. (n.d.). Word Embedding. Retrieved from https://dataanalyticspost.com/Le embedding.

[40] Turing. (n.d.). A Guide on Word Embeddings in NLP. Retrieved from https://www.turing.com/kb/guide-on-word-embeddings-in-nlp.

[41] Mikolov, T., et al. (2013). Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems, 26.

[42] Babić, K., Guerra, F., Martinčić-Ipšić, S., and Meštrović, A. (2020). A Comparison of Approaches for Measuring the Semantic Similarity of Short Texts Based on Word Embeddings. JIOS, 44(2).

[43] Hashimoto, T. B., Alvarez-Melis, D., and Jaakkola, T. S. (Year). Word Embeddings as Metric Recovery in Semantic Spaces. CSAIL, Massachusetts Institute of Technology.

[44] Shoham, S. (2023). What are word vectors. Retrieved from https://www.kubiya.ai/resource-post/what-are-word-vectors.

[45] Rong, X. (2016). word2vec Parameter Learning Explained. arXiv:1411.2738v4 [cs.CL].

[46] Ruder, S. (2016). On word embeddings - Part 3: The secret ingredients of word2vec. Retrieved from https://www.ruder.io/secret-word2vec/.

[47] Xia, H. (2023). Continuous-bag-of-words and Skip-gram for word vector training and text classification. CONF-CIAP 2023.

[48] Pennington, J., Socher, R., and Manning, C. D. (2014). GloVe: Global Vectors for Word Representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP).

[49] Bullinaria, J. A., and Levy, J. P. (2007). Extracting semantic representations from word co-occurrence statistics: A computational study. Behavior Research Methods, 39(3), 510-526.

[50] Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2016). FastText: Zipfian word representations. arXiv preprint arXiv:1603.05201.

[51] Analytics Vidhya. (2024). Introduction to FastText Embeddings and its Implication. Retrieved from https://www.analyticsvidhya.com.

[52] Mikolov, T., et al. (2013). Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems, 26.

[53] Vaswani, A., et al. (2017). Attention is All You Need. arXiv:1706.03762. Retrieved from https://arxiv.org/pdf/1706.03762.

[54] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... and Polosukhin, I. (2017). Attention is All You Need. In Advances in Neural Information Processing Systems (NeurIPS) (pp. 5998-6008).

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

[56] Joulin, A., et al. (2016). FastText: Zipfian word representations. arXiv:1603.05201. Retrieved from https://arxiv.org/abs/1603.05201.

[57] Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2016). Bag of Tricks for Efficient Text Classification. arXiv:1603.05201.

[58] Yu, Lei, et al. (2018). A decoder-only transformer for sequence-to-sequence learning. arXiv:1808.08444. Retrieved from https://arxiv.org/abs/1808.08444.

[59] Joulin, A., et al. (2016). Bag of tricks for efficient text classification. In Proceedings of the 15th Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics.

[60] Howard, Jeremy, and Sebastian Ruder. (2018). Universal language model fine-tuning for text classification. arXiv:1801.06146. Retrieved from https://arxiv.org/abs/1801.06146.

[61] Joulin, A., et al. (2016). Bag of tricks for efficient text classification. In Proceedings of the 15th Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics.

[62] Yu, Lei, et al. "Quality injection for machine translation." arXiv preprint arXiv:1804.09791, 2018.

[63] Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." Advances in Neural Information Processing Systems, 2013.

[64] Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." Advances in Neural Information Processing Systems, 2013.

[65] Pennington, Jeffrey, et al. "GloVe: Global vectors for word representation." arXiv preprint arXiv:1406.1075, 2014.

[66] Socher, Richard, et al. "Recursive deep learning for semantic sentiment analysis." arXiv preprint arXiv:1306.6086, 2013.

[67] Tang, Duyu, et al. "A sentiment analysis system based on machine learning." Proceedings of the 2009 International Conference on Computational Intelligence and Natural Computing, Vol. 5. IEEE, 2009.

[68] Giorcelli, Michela, Nicola Lacetera, Astrid Marinoni. "How does scientific progress affect cultural changes? A digital text analysis." April 2022, 27(3):1-38.

[69] Joulin, Armand, et al. "FastText: Zipfian word representations." arXiv preprint arXiv:1603.05201, 2016.

[70] Chiu, John PC, and James CR. Manning. "Named entity recognition with bidirectional LSTM-CNNs." arXiv preprint arXiv:1507.07003, 2015.

[71] Lample, Guillaume, et al. "Conditional random fields for jointly learning segmentation and labeling of task-specific sequences." Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pp. 1-11, 2013.

[72] Nguyen, Thanh, et al. "Joint named entity recognition and relation classification." Proceedings of the 2016 Conference on North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 858-868, 2016.

[73] Vaswani, Ashish, et al. "Attention is all you need." Advances in Neural Information Processing Systems 31, 2017, pp. 5995-6005.

[74] Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." Advances in neural information processing systems, 2013.

[75] Pennington, Jeffrey, et al. "GloVe: Global vectors for word representation." arXiv preprint arXiv:1406.1075, 2014.

[76] Dr. Claudio Calvino, Nathalie Baker, Dr. Dimitris Korres, February 26, 2024. "Machine Learning Model Metrics – Can I Trust Them?" Available at: https://www.fticonsulting.com/ins model-metrics-trust-them.

[77] Fidler, Sandra, et al. "Semantic parsing for image understanding." European Conference on Computer Vision. Springer, Berlin, Heidelberg, 2013.

[78] Hill, Felicity, et al. "Exemplar similarity based word representation." arXiv preprint arXiv:1509.08830, 2015.

[79] Bolukbasi, Tolga, et al. "Man is to woman as computer is to home: Debiasing word embeddings." Advances in Neural Information Processing Systems 30, 2016, pp. 2214-2223.

Chapter 3:

[80] Hawkins, D. M. (1980). Identification of outliers (Vol. 11). Chapman and Hall/CRC.

[81] Cortes, C., and Vapnik, V. (1995). Support-vector networks. Machine Learning, 20(3), 273-297.

[82] Kim, S. H., and Lee, H. G. (2017). Big data analytics and firm performance: Findings from South Korea. Telecommunications Policy, 41(10), 948-961.

[83] Westerman, G., Bonnet, D., and McAfee, A. (2014). Leading digital: Turning technology into business transformation. Harvard Business Press.

[84] Ramakrishnan Kannan Abstract Hyenkyun Woo Charu C. Aggarwal Haesun Park: Outlier Detection for Text Data Downloaded 05/09/24 to 105.104.95.201 . Available at: https://epubs.siam.org/terms-privacy.

[85] Kay Liu, Yingtong, Yue Zhao, Liu et al. (2018). BOND: Benchmarking Unsupervised Outlier Node Detection on Static Attributed Graphs.

[86] Bishop, C. M. (2006). Pattern recognition and machine learning. Springer.

[87] Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. ACM Computing Surveys (CSUR), 41(3), 1-58.

[88] Kusner, M. J., Sun, Y., Kolkin, N. I., and Weinberger, K. Q. (2015). From word embeddings to document distances. In Proceedings of the 32nd International Conference on Machine Learning (ICML) (Vol. 37, pp. 957-966).

[89] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. Advances in Neural Information Processing Systems, 26, 3111-3119.

[90] Bird, S., Klein, E., and Loper, E. (2009). Natural language processing with Python: Analyzing text with the natural language toolkit. O'Reilly Media, Inc.

[91] Manning, C. D., Raghavan, P., and Schütze, H. (2008). Introduction to information retrieval. Cambridge University Press.

[92] Kusner, M. J., Sun, Y., Kolkin, N. I., and Weinberger, K. Q. (2015). From word embeddings to document distances. Proceedings of the 32nd International Conference on Machine Learning (ICML-15), 37, 957-966.

[93] Word Mover's Distance. Retrieved from https://radimrehurek.com/gensim/auto*examples/tutorials/r*

[94] Kusner, M. J., Sun, Y., Kolkin, N. I., and Weinberger, K. Q. (2015). From word embeddings to document distances. In Proceedings of the 32nd International Conference on Machine Learning (ICML) (Vol. 37, pp. 957-966).

[95] Aggarwal, C. C., Hinneburg, A., and Keim, D. A. (2001). On the surprising behavior of distance metrics in high dimensional space. In International Conference on Database Theory (pp. 420-434). Springer, Berlin, Heidelberg.

[96] Manning, C. D., Raghavan, P., and Schütze, H. (2008). Introduction to information retrieval. Cambridge University Press.

[97] Manning, C. D., Raghavan, P., and Schütze, H. (2008). Introduction to information retrieval. Cambridge University Press.

[98] Spencer Porter. Understanding Cosine Similarity and Word Embeddings. Available at: https://spencerporter2.medium.com/understanding-cosine-similarity-and-wordembeddings-dbf19362a3c.

[99] Matt J. Kusner, Yu Sun, Nicholas I. Kolkin, Kilian Q. Weinberger. (2015). From Word Embeddings To Document Distance.

[100] Empirical Performance Often shows superior performance in benchmarks involving semantic similarity tasks, such as STS (Semantic Textual Similarity) benchmarks.

[101] Salton, G., Wong, A., and Yang, C. S. (1975). A vector space model for automatic indexing. Communications of the ACM, 18(11), 613-620.

[102] Pele, O., and Werman, M. (2009). Fast and robust earth mover's distances. In 2009 IEEE 12th International Conference on Computer Vision (pp. 460-467). IEEE.

[103] Singhal, A. (2001). Modern information retrieval: A brief overview. IEEE Data Engineering Bulletin, 24(4), 35-43.

[104] Goutte, C., and Gaussier, E. (2005). A probabilistic interpretation of precision, recall and F-score, with implications for evaluation. In European Conference on Information Retrieval (pp. 345-359). Springer.

[105] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

[106] Agirre, E., Cer, D., Diab, M., and Gonzalez-Agirre, A. (2012). SemEval-2012 task 6: A pilot on semantic textual similarity. In Proceedings of the First Joint Conference on Lexical and Computational Semantics (SEM) (pp. 385-393).

[107] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. Advances in Neural Information Processing Systems, 26, 3111-3119.

[108] Kusner, M. J., Sun, Y., Kolkin, N. I., and Weinberger, K. Q. (2015). From word embeddings to document distances. In Proceedings of the 32nd International Conference on Machine Learning (ICML) (Vol. 37, pp. 957-966).

[109] Le, Q., and Mikolov, T. (2014). Distributed representations of sentences and documents. In International Conference on Machine Learning (pp. 1188-1196).

[110] Tang, D., Qin, B., and Liu, T. (2015). Learning semantic representations of users and products for document level sentiment classification. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers) (Vol. 1, pp. 1014-1023).

[111] Artetxe, M., Labaka, G., and Agirre, E. (2018). A robust self-learning method for fully unsupervised cross-lingual mappings of word embeddings. arXiv preprint arXiv:1711.00284.

[112] Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. arXiv preprint arXiv:1802.05365.

[113] Word Mover's Embedding: From Word2Vec to Document Embedding. arXiv:1811.01713v1 [cs.CL] 30 Oct 2018.

[114] Serkan Ballı, Onur Karasoy. Development of content-based SMS classification application by using Word2Vec-based feature extraction. E-First on 11th December 2018.

[115] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching Word Vectors with Subword Information. Transactions of the Association for Computational Linguistics, 5, 135-146.

[116] Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., and Joulin, A. (2018). Advances in Pre-Training Distributed Word Representations. arXiv preprint arXiv:1712.09405.

[117] Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2017). Bag of Tricks for Efficient Text Classification. arXiv preprint arXiv:1607.01759.

[118] Amit Chaudhary (June 21, 2020). A Visual Guide to FastText Word Embeddings. Available at: https://amitness.com/posts/fasttext-embeddings.

[119] Sasaki, Y. (2007). The truth of the F-measure. Teach Tutor Mater, 1(5), 1-5.

[120] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. arXiv preprint arXiv:1301.3781. Retrieved from https://arxiv.org/abs/1301.3781.

[121] Kusner, M. J., Sun, Y., Kolkin, N. I., and Weinberger, K. Q. (2015). From Word Embeddings to Document Distances. In Proceedings of the 32nd International Conference on Machine Learning (ICML-15), 957-966.

[122] Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2016). Bag of Tricks for Efficient Text Classification.

[123] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching Word Vectors with Subword Information.

[124] Python Software Foundation. (n.d.). Tkinter - Python interface to Tcl/Tk. Available at: https://docs.python.org/3/library/tkinter.html.