Duale Hochschule Baden-Württemberg Mannheim

Seminararbeit

Identifizierung von möglichen Sanktionierten Geschäftspartnern mithilfe von fuzzy search in Python

Studiengang Wirtschaftsinformatik

Studienrichtung Data Science

Verfasser(in):	Joshua Brenzinger, Luis Steinert, Pascal Breucker
Matrikelnummern:	1960679, 2617416, 5800129
Kurs:	WWI20DSA
Studiengangsleiter:	Dennis Pfisterer
Wissenschaftliche(r) Betreuer(in):	Maximilian Scherer
Bearbeitungszeitraum:	21.11.2022 – 10.02.2023

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Es gibt mehrere Gründe, warum es wichtig ist, herauszufinden, ob ein potenzieller Geschäftspartner auf einer Sanktionsliste steht.

1. **Einhaltung der Vorschriften:** Viele Länder und Organisationen haben Gesetze und Vorschriften erlassen, die es Einzelpersonen oder Unternehmen verbieten, Geschäfte mit Einrichtungen zu tätigen, die auf einer Sanktionsliste stehen. Die Nichteinhaltung dieser Gesetze kann schwere Strafen nach sich ziehen, darunter Geld- und Haftstrafen.
2. **Ruf:** Die Verbindung mit einem Unternehmen oder einer Person, die auf einer Sanktionsliste steht, kann den Ruf eines Unternehmens schädigen und zu Geschäftseinbußen und negativer Publicity führen.
3. **Risiko:** Die Zusammenarbeit mit einem Unternehmen oder einer Person, die auf einer Sanktionsliste steht, kann ein Unternehmen finanziellen und rechtlichen Risiken aussetzen, z. B. der Beschlagnahme von Vermögenswerten, dem Einfrieren von Bankkonten und Strafen für Sanktionsverstöße.
4. **Nationale Sicherheit:** Einige Unternehmen auf der Sanktionsliste stehen aus Gründen der nationalen Sicherheit dort. Daher ist es für Unternehmen wichtig, ihre Geschäftspartner zu überprüfen, um sicherzustellen, dass sie nicht in Aktivitäten verwickelt sind, die den nationalen Sicherheitsinteressen schaden könnten. Insgesamt ist die Feststellung, ob ein potenzieller Geschäftspartner auf einer Sanktionsliste steht, ein entscheidender Schritt zum Schutz der finanziellen und rechtlichen Interessen eines Unternehmens sowie seines Rufs und seiner möglichen Verwicklung in Handlungen, die der nationalen Sicherheit schaden könnten.

Auf Basis dieser Thematik beschreibt diese Arbeit das Vorgehen beim Entwickeln eines Screening-Algorithmus, unter Verwendung existierender Python-Bibliotheken zum finden dieser "Business-Partner". Dabei sollen Methoden des Word-Embedding sowie das "fuzzy"-Suchen dieser Personen und Unternehmen (Namen und Adresse) vergleichend Analysiert und basierend auf Gütekriterien bewertet werden. Diese Qualitätskennzahlen zusammen mit der Suchfunktion wird über ein per Streamlit entwickeltes User Interface, dargestellt werden. Abgrenzend wird erwähnt, dass dabei nicht die Zugehörigkeit zu etwaigen Terrororganisation oder ähnlichem angezeigt wird.

2 Datengrundlage

2.1 Grundlagen der European Sanctions List

Die European Sanctions List - European Union Consolidated Financial Sanctions List – enthält die konsolidierte Liste der Personen, Vereinigungen und Körperschaften, deren Vermögenswerde nach den EU-Sanktionen eingefroren werden und denen es untersagt ist, Gelder und wirtschaftliche Ressourcen zur Verfügung stellen.

Grundsätzlich sind die Listen der für die einzelnen Sanktionsprogramme benannten Personen und Einrichtungen den jeweiligen EU-Ratsbeschlüssen und Durchführungsverordnungen beigefügt und liegen getrennt voneinander vor. Um die Anwendung von Finanzsanktionen zu erleichtern, erkannten die Europäische Bankenvereinigung, die Europäische Sparkassenvereinigung, die Europäische Vereinigung der Genossenschaftsbanken und die Europäische Vereinigung öffentlicher Banken jedoch die Notwendigkeit einer konsolidierten EU-Liste der Personen, Gruppen und Einrichtungen, die Finanzsanktionen im Zusammenhang mit der Gemeinsamen Außen- und Sicherheitspolitik der EU unterliegen.

Die konsolidierte europäische Liste der Finanzsanktionen dient als Datengrundlage beziehungsweise als Corpus für die weitere Auswertung des Projekts (vgl. EuropeanCommission, 2020).

2.2 Preprocessing

Das Preprocessing bezieht sich auf den Prozess, Daten vor einer Analyse oder Verarbeitung vorzubereiten. Die genaue Vorgehensweise hängt von den spezifischen Anforderungen und den Daten ab, wobei einige Schritte grundlegend vorzunehmen sind.

Zieldefinition: Ausgehend von der Sanktionsliste muss eine Einteilung für jeden Eintrag erfolgen und darüber hinaus jegliche angegebenen Namen, sowie deren zugeordnete Kontaktinformationen separat dargestellt werden.

Datenaufnahme: Die konsolidierte Liste der Finanzsanktionen liegt als PDF-Datei vor. Um im Rahmen des Projekts Zugriff zu erhalten, ist diese im ersten Schritt einzulesen.

Das Python-Modul PyPDF2 ermöglicht die zeilenweise Übernahme des Inhalts, sowie die Bearbeitung und Erstellung von PDF-Dateien. Eine alternative Herangehensweise stellt die Extraktion der Daten mittels Image-Based Methoden dar. Optical Character Recognition, zum Beispiel die open-source OCR engine Tesseract, ermöglicht die Extraktion von Text aus Bildern oder PDF-Dateien,

Datenaufbereitung- und bereinigung: Die Datenaufbereitung bezieht sich im ersten Schritt auf die Transformation des eingelesenen Textes in einen Dataframe. Der Dataframe soll die folgenden Spalten enthalten EU Reference Number, Legal Basis, Programme, Identity Information, Citizenship Information, Contact Information. Eine Unterteilung des Textes in die benannten Personen und Entitäten erfolgt anhand der EU Reference Number, welche einen neuen Eintrag darstellt. Die Spalte Identity Information des Dataframes kann mehre Namen, sowohl Vornamen als auch Nachnamen, und Rollen der jeweiligen Person enthalten. Im Rahmen der Datenbereinigung ging ist mehrheitlich um den Inhalt der definierten Spalten. Es galt eine Unterteilung der verschiedenen eingetragenen Namen und bereinigung von Textinhalten, die einen negativen Einfluss auf die Suche nehmen.

2.3 Herausforderungen

Grundsätzlich ist eine Unterteilung der Sanktionseinträge anhand der EU Reference Number, welche jede Person beziehungsweise Entität betitelt, möglich. Darüber hinaus werden jedoch weitere Informationen unstrukturiert dargestellt. Die Angaben in Bezug auf die Identity Informationen der Personen können beispielweise einzelne Namen, sowie Vor- und Nachnamen oder auch mehrere Einträge sein. Zusätzlich sind diese Teilweise durch Funktionen der Personen und vollständige Sätze ergänzt. Im Fall von Entitäten, kann die Spalte Identity Information weiterhin das Feld, in dem das Unternehmen tätig ist, enthalten. Die Identity Information ist jedoch für alle Einträge grundsätzlich gegeben. Weitere Informationen in Bezug auf die Herkunft, die Kontaktinformationen oder weitere Anmerkungen der Personen oder Entitäten sind fehlerhaft oder unvollständig und können verschiedene Inhalte haben.

3 Word Embedding

3.1 Abgrenzung der Methode

Word Embedding ist eine Technik in der maschinellen Sprachverarbeitung, bei der jedem Wort eines Textes ein numerischer Vektor (Embedding) zugewiesen wird, um seine Bedeutung und Kontext innerhalb des Textes zu repräsentieren. Diese Vektoren können dann in maschinellen Lernmodellen verwendet werden, um Aufgaben wie Sentiment-Analyse, Klassifikation und Übersetzung auszuführen. Die grundlegende Idee von Word Embedding ist es, die Bedeutung von Wörtern durch ihre Verwendung und Zusammenhang in großen Textmengen zu erfassen und in einer numerischen Form zu repräsentieren (vgl. Almeida und Xexéo, 2019). In der Regel werden für Word Embeddings, Word2Vec Modelle eingesetzt um auf einen großen Corpus mit Text, Wörter und deren Zusammenhänge mittels Modelltraining identifizieren zu können. Während des Trainings wird das Modell angepasst, um eine möglichst gute Vorhersage für das nächste Wort im Kontext des aktuellen Wortes und seines Embeddings zu treffen. Nach Abschluss des Trainings können die Embeddings für jedes Wort im Textkorpus extrahiert werden und in weiteren maschinellen Lernmodellen verwendet werden.

3.2 Word2Vec

Das Ziel von Word2Vec ist es, für jedes Wort in einem Textkorpus einen Vektor zu lernen, der seine Bedeutung und seinen Kontext in Beziehung zu anderen Wörtern beschreibt. Das Modell lernt diese Vektoren, indem es den Kontext eines Wortes vorhersagt, wenn es dessen Bedeutung kennt. Dies wird durch die Verwendung eines neuronalen Netzes erreicht, das eine sogenannte Skip-Gram-Architektur verwendet. Das Modell besteht aus einem Input-Layer, einem hidden Layer und einem Output-Layer. Die Input-Schicht enthält das aktuelle Wort, das durch seinen Vektor repräsentiert wird. Die versteckte Schicht enthält eine feste Anzahl von Neuronen und stellt die Bedeutung des aktuellen Wortes dar. Die Output-Schicht enthält so viele Neuronen wie es eindeutige Wörter im Textkorpus gibt und repräsentiert die vorhergesagten Kontextwörter (vgl. Church, 2017).

3.3 Cosine Similarity

Um die Ähnlichkeit zwischen zwei nicht-null Vektoren eines inneren Produktraumes zu spezifizieren, wird die "Cosine Similarity" der Winkel verwendet, indem die beiden Vektoren zueinander stehen. Grundlegend benötigt man für den mathematischen Hintergrund die Definition des Vektors d und Eingabe q. Dabei wird die Summe der Multiplikation der Werte von q und d durch die Wurzel aus der Summe der Werte von q multipliziert, mit der Wurzel aus der Summe der Werte von q, dividiert. Die Gleichheit der Vektoren d und q kann somit mittels nachstehender Formel definiert werden.

$$Sim(\vec{q}, \vec{d}) = \frac{\sum_k^t w_q k * w_d k}{\sqrt{(\sum_k^t (w_q k)^2)} * \sqrt{(\sum_k^t (w_d k)^2)}}$$

Nachteile

Nachteile sind im allgemeinen, dass mit „Cosine Similarity" der semantische Zusammenhang und Kontext zwischen den untersuchten Wordvektoren nicht immer zum gewünschten perfekten Ergebnis führt. Aushilfe verschaffen könnte die Verwendung eines Lexikons wie etwa "WordNet", wobei der Kontext basierend auf menschlich erzeugter Evaluation analysiert und verwendet werden kann. (vgl. Rahutomo et al., 2012).

3.4 Erklärung der nicht Umsetzbarkeit

In diesem Kontext macht es keinen Sinn, Word Embedding mit Word2Vec zu verwenden, da Word2Vec für die semantische Analyse von Wörtern in großen Textkorpora entwickelt wurde. Im Gegensatz dazu geht es bei der Identifizierung von Geschäftspartnern in einem PDF-Dokument jedoch um eine einfache String-Suche. Es ist nicht notwendig, semantische Beziehungen zwischen den Wörtern zu analysieren, sondern lediglich darum, ob ein bestimmter String im Text vorhanden ist. In diesem Fall ist eine einfache String-Suche oder eine Suchalgorithmus wie Fuzzy Search ausreichend, um die Namen der Geschäftspartner zu finden. Die Verwendung von Word Embedding mit Word2Vec wäre in diesem Kontext überflüssig und würde nur unnötige Komplexität hinzufügen. Auf Basis dieser Erkentniss, wurde das Projekt im weiteren Verlauf mithilfe von Named Entity Recognition durchgeführt, um weiter Preprocessing Schritte durchführen zu können.

4 Named Entity Recognition

4.1 Einordnung NER

Die Erkennung von benannten Entitäten (Named Entity Recognition, NER) ist eine Teilaufgabe der natürlichen Sprachverarbeitung (Natural Language Processing, NLP), bei der es um die Identifizierung und Kategorisierung von benannten Entitäten geht, die in einem Textdokument erwähnt werden, z. B. Personennamen, Organisationen, Orte, medizinische Codes, Zeitangaben, Mengen, Geldwerte, Prozentsätze usw.

Das Ziel von NER ist es, strukturierte Informationen aus unstrukturierten Textdaten zu extrahieren und sie in ein maschinenlesbares Format zu konvertieren, das dann für verschiedene NLP-Anwendungen wie Information Retrieval, Fragebeantwortung, Textklassifizierung, maschinelle Übersetzung und viele andere verwendet werden kann.

Es gibt verschiedene Ansätze für NER, darunter regelbasierte, wörterbuchbasierte und statistische Methoden. Regelbasierte Methoden stützen sich auf manuell erstellte Regeln, um benannte Entitäten zu identifizieren, wohingegen wörterbuchbasierte Methoden ein vordefiniertes Wörterbuch oder einen Gazetteer verwenden, um die Entitäten abzugleichen. Statistische Methoden hingegen lernen von annotierten Trainingsdaten und können in zwei weitere Kategorien unterteilt werden: traditionelle, auf maschinellem Lernen basierende Methoden und auf Deep Learning basierende Methoden. Auf Deep Learning basierende NER-Modelle, wie rekurrente neuronale Netze (RNNs), Faltungsneuronale Netze (CNNs) und Transformatoren, haben bei verschiedenen NER-Datensätzen Spitzenleistungen erzielt und sind zur bevorzugten Methode für NER geworden. Diese Modelle können kontextuelle Informationen und semantische Beziehungen zwischen Wörtern in einem Satz effektiv erfassen, was zu einer besseren NER-Leistung im Vergleich zu traditionellen, auf maschinellem Lernen basierenden Methoden führt. Zusammenfassend lässt sich sagen, dass NER eine wichtige Aufgabe in der NLP ist, die dabei hilft, strukturierte Informationen aus unstrukturierten Textdaten zu extrahieren, und weitreichende Anwendungen in verschiedenen Bereichen wie Information Retrieval, Fragebeantwortung und Textklassifikation hat.

4.2 Motivation für den Einsatz von Namenserkennung

Ein vorab trainiertes Modell, das häufig für die Erkennung von benannten Entitäten verwendet wird und für die Identifizierung von Namen in einem Text nützlich sein kann, ist das BERT-Modell (Bidirectional Encoder Representations from Transformers). BERT ist ein transformatorbasiertes Deep-Learning-Modell, das bei einer Vielzahl von NLP-Aufgaben, einschließlich NER, gute Leistungen erzielt hat.

BERT wird auf großen Mengen von Textdaten trainiert und kann so umfangreiche Kontextinformationen und Beziehungen zwischen Wörtern in einem Satz lernen und erfassen. Diese Kontextinformationen können für die genaue Identifizierung von Namen in einem Text nützlich sein, da Namen oft kontextuelle Abhängigkeiten mit den sie umgebenden Wörtern aufweisen.

Ein weiteres vortrainiertes Modell, das häufig für NER verwendet wird, ist das spaCy NER-Modell. spaCy ist eine Open-Source-NLP-Bibliothek, die ein vortrainiertes NER-Modell enthält, das für bestimmte NER-Aufgaben fein abgestimmt werden kann. Das spaCy-NER-Modell basiert auf einem Deep-Learning-Ansatz und kann für eine Vielzahl von NER-Aufgaben verwendet werden, einschließlich der Identifizierung von Namen in einem Text.

Zusammenfassend lässt sich sagen, dass sowohl das BERT- als auch das spaCy-NER-Modell leistungsstarke vortrainierte Modelle sind, die für die Identifizierung von Namen in einem Text nützlich sein können. BERT hat bei NER-Aufgaben eine Spitzenleistung erzielt und verfügt über eine starke Fähigkeit zur Erfassung von Kontextinformationen, während spaCy NER ein vielseitiges Modell ist, das für spezifische NER-Aufgaben fein abgestimmt werden kann.

4.3 Grenzen für Use Case deutlich machen

Bei der Verwendung eines vortrainierten Named-Entity-Recognition-Modells zur Identifizierung von Namen aus verschiedenen Kulturen in einem Textkorpus sind mehrere Einschränkungen zu beachten:

- Sprachliche und kulturelle Verzerrungen: Vorgefertigte NER-Modelle werden auf großen Mengen von Textdaten trainiert, die oft die sprachlichen und kulturellen Vorlieben

der Datenquelle widerspiegeln. Wenn der Textkorpus Namen aus verschiedenen Kulturen enthält, die in den Trainingsdaten nicht gut vertreten sind, kann das Modell Schwierigkeiten haben, diese Namen genau zu erkennen und zu kategorisieren.

- Abdeckung des Vokabulars: NER-Modelle verwenden ein vordefiniertes Vokabular von benannten Entitäten, um benannte Entitäten in einem Text zu identifizieren und zu kategorisieren. Wenn das Vokabular keine Namen aus verschiedenen Kulturen enthält, ist das Modell nicht in der Lage, diese Namen zu erkennen, selbst wenn sie im Textkorpus gut vertreten sind.
- Nicht standardisierte Benennungskonventionen: Verschiedene Kulturen können unterschiedliche Namenskonventionen haben, wie z. B. die Verwendung von Zweitnamen oder mehreren Nachnamen, was für NER-Modelle, die auf westliche Namenskonventionen trainiert sind, eine Herausforderung darstellen kann. Das Modell kann auch Schwierigkeiten haben, Namen zu erkennen, die Zeichen oder Symbole enthalten, die nicht Teil des Standard-ASCII-Zeichensatzes sind.
- Mehrdeutigkeit: Namen aus verschiedenen Kulturen können in verschiedenen Kontexten unterschiedliche Bedeutungen haben, so dass es für ein NER-Modell schwierig ist, die richtige Bedeutung ohne zusätzlichen Kontext zu bestimmen. Der Name "Lee"könnte sich beispielsweise auf einen Nachnamen oder einen Vornamen beziehen, und das Modell könnte Schwierigkeiten haben, die korrekte Interpretation zu bestimmen.

Programme: A ORG FG - Afghanistan LOC Identity information: • Name/Alias: Abdul Baqi Basir Awal Shah PER Title: (a) Ma PER ulavi, (b) Mu PER llah Function: (a) Governor of Khost LOC and Paktika LOC provinces under the Taliban MISC regime, (b) Vice-Minister of Information and Culture ORG under the Taliban ORG regime, (c) Consulate Department, Ministry of Foreign Affairs ORG under the Taliban ORG regime. • Name/Alias: Abdul Baqi PER Birth information: • Birth date: Circa LOC from 1960 to 1962 Birth place: Afghanistan LOC , Shinwar District LOC , Nangarhar Province LOC • Birth date: from 1960 to 1962 Birth place: Afghanistan LOC , Jalalabad City LOC , Nangarhar Province LOC Citizenship information: • Citizenship: Afghanistan LOC

Abbildung 4.1: Exemplarisches Ergebniss von NER durch base_bert_NER

Das Modell base_bert_NER „base_bert_NER“, n. d. kann vier verschiedenen Entitäten erkennen (PER, LOC, ORG, MISC). Es ist bei diesem exemplarischen Output von base_bert_NER zu erkennen, dass Orte (LOC) perfekt klassifiziert werden. Dies ist mit hoher Wahrscheinlichkeit auf die einheitliche Schreibweise bei Orten zurückzuführen, welche in der Consolidated List of Financial Sanctions der EU zur Anwendung kommt. Bei Namen fällt auf, dass die Klassifizierung nicht konsistent ist. In der zweiten Zeile ist zu sehen, dass nur bestimmte Silben der Namen als Person (PER) erkannt werden. Dies wäre für den Use Case hier sehr problematisch, da man so den Datensatz um potenziell relevante Informationen verkürzt und so die Möglichkeit eröffnet wird, sanktionierte Personen, die in dem Datensatz vorkommen, zu löschen.

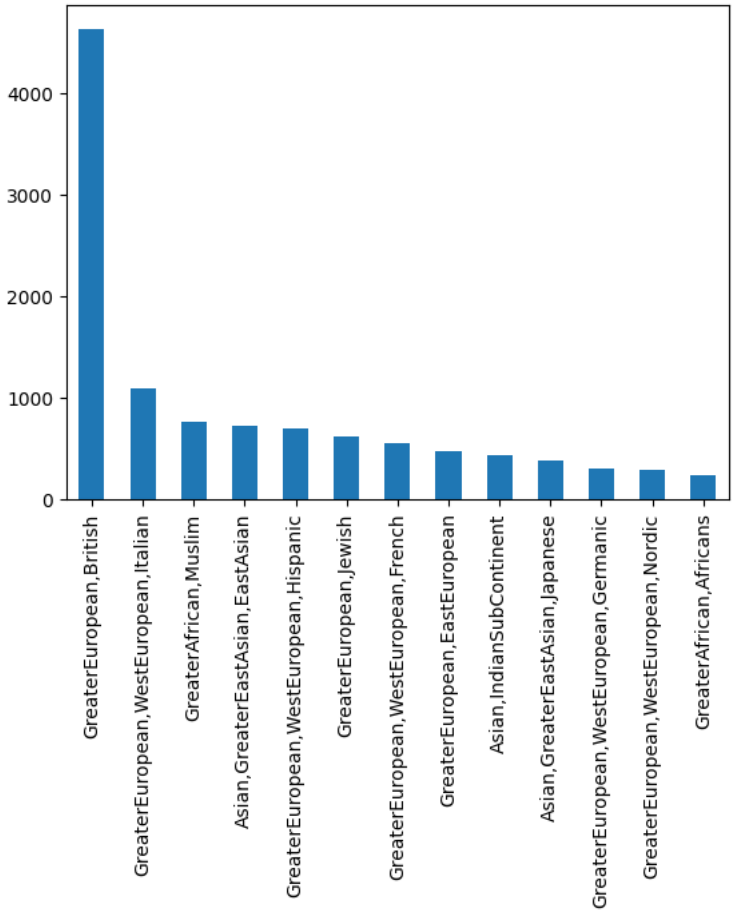

Abbildung 4.2: Barplot mit Verteilung der Regionen der Namen auf Grundlage des Pakets ethnicolr

Ein möglicher Grund für die Güte des Modells bei dem Tag "Person"könnte eine nicht ausbalancierte Datengrundlage sein. Dies lässt sich bestätigen, wenn man sich die Datengrundlage, welche für das Finetuning des base_bert Modells verwendet wurde. Der

CoNLL-2003 beruht auf Artikeln der Nachrichtenagentur Reuters zwischen dem August 1996 und August 1997. In dem Datensatz, der für das Training verwendet wurde, sind $\sim$ 200.000 Tokens vorhanden, von denen $\sim$ 6000 mit dem Tag "Person"versehen wurden. In der folgenden Grafik wird mithilfe des Python Packages dargstellt, aus welchen Regionen, die für das Training der "PER"Klasse verwendeten Namen, stammen.

Hier lässt sich erkennen, dass es eine starke Asymmetrie des Datensatzen gibt. Es ist davon auszugehen, dass Namen aus Regionen, die weniger in dem Trainingsdatensatzes vorkommen, demnentsprechend auch schlechter erkannt werden.

Zusammenfassend lässt sich sagen, dass die Verwendung eines vortrainierten NER-Modells zur Identifizierung von Namen aus verschiedenen Kulturen in einem Textkorpus aufgrund von sprachlichen und kulturellen Verzerrungen, begrenzter Vokabelabdeckung, nicht standardisierten Namenskonventionen und Mehrdeutigkeit eine Herausforderung darstellen kann. Um diese Einschränkungen zu überwinden, muss das Modell möglicherweise an einem Korpus, das für die Zielsprache und -kultur repräsentativer ist, feinabgestimmt werden, oder es müssen zusätzliche Kontextinformationen in das Modell aufgenommen werden, um seine Genauigkeit zu verbessern.

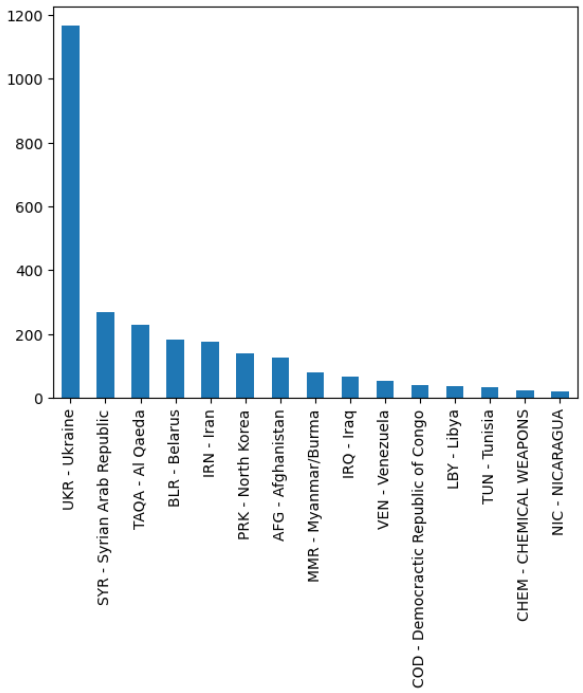

Abbildung 4.3: Barplot mit Verteilung der Vorkommen der Programme in EU Consolidated Financial List

5 Fuzzy String Matching

5.1 Problemstellung schlagwortbasierter Internet-Suchmaschinen

Das Konzept des Fuzzy String Matching, um dahingehend Fuzzy Search zu ermöglichen, entstand als Lösung eines der ursprünglichen Probleme von Internet-Suchmaschinen. Da die Datenmenge im Internet rapide zugenommen hat, wurden Internet-Suchmaschinen zu einem unverzichtbaren Mittel für die Informationssuche. Die meisten bestehenden Suchmaschinen wie Google, Yahoo, MSN, ASK, Baidu und Bing riefen Webseiten durch die Suche nach genauen Schlüsselwörtern ab. Diese schlagwortbasierten Suchmaschinen sammeln und analysieren Webseiten mit Hilfe von Webcrawlern.

Während die Benutzer Schlüsselwörter eingeben, um Webseiten zu suchen, werden Webseiten, die exakte Schlüsselwörter enthalten, abgerufen und in eine Rangfolge gebracht. Die Google-Suchmaschine zum Beispiel sortiert die Suchergebnisse nach den Kriterien Page Rank Score, Relevance Score und Local Scores (vgl. Lai et al., 2011). Lai et al. schlugen als erste ein Fuzzy-Schlüsselwort-Suchschema für verschlüsselte Daten vor. Sie nutzten die verschiedene Distanzen, um die Ähnlichkeit von Schlüsselwörtern zu qualifizieren und entwickelten eine Technik zur Konstruktion von Fuzzy-Schlüsselwortmengen und führten dabei grundlegende Probleme der Suchmaschinen auf.

Die Probleme, die sich für schlagwortbasierte Internet-Suchmaschinen ergeben, sind gleichermaßen gültig für Suchalgorithmen, die innerhalb eines gegebenen Strings, einer Liste oder ähnliches einen bestimmten Begriff suchen und können somit auf die Namenssuche von Business Partners innerhalb der European Sanctions List übertragen werden. Grundsätzlich muss mit folgenden Problemen umgegangen werden.

- Synonyme und Begriffe, die den Schlüsselwörtern ähnlich sind, werden bei der Suche auf Webseiten nicht berücksichtigt. Die Benutzer müssen möglicherweise mehrere ähnliche Schlüsselwörter einzeln eingeben, um eine Suche abzuschließen. Die Beschränkung auf exakte Schlüsselwörter macht die Suche nach Webseiten für die

Nutzer umständlich. Viele wertvolle Webseiten würden übersehen, wenn die Benutzer nicht nach mehreren ähnlichen Schlüsselwörtern einzeln suchen würden (vgl. Lai et al., 2011).

- Während die Benutzer mehrere Schlüsselwörter eingeben, um Webseiten zu durchsuchen, können die verschiedenen Schlüsselwörter in ihrer Meinung unterschiedlich wichtig sein. Herkömmliche Suchmaschinen behandeln alle Schlüsselwörter als gleich wichtig und können die Bedeutung eines Schlüsselworts nicht von der eines anderen unterscheiden (vgl. Lai et al., 2011).
- Das Problem der Informationsüberflutung macht es den Nutzern schwer, aus einer großen Menge von Suchergebnissen wirklich nützliche Informationen zu finden. Herkömmlichen Suchmaschinen fehlt ein geeigneter Klassifizierungsmechanismus, um den Suchraum zu verkleinern und die Suchergebnisse zu verbessern (vgl. Lai et al., 2011).

Fuzzy String Matching beziehungsweise Fuzzy-Search soll folgend eingesetzt werden, um diesen Problemen teilweise entgegenzuwirken. Die Fuzzy-Search ist eine unscharfe oder fehlertolerante Suche, die der Klasse von String-Matching-Algorithms zuzuordnen ist. Im Gegensatz zu einer exakten Suche werden bei der Fuzzy-Suche auch dann Ergebnisse angezeigt, wenn das Suchwort fehlerhaft eingegeben wurde oder wenn die exakte Schreibweise des Suchwortes nicht bekannt ist. Als Suchkriterium wird somit nicht die exakte Zeichenfolge zugrunde gelegt.

Der unscharfe Suchalgorithmus verwendet für den Abgleich von Strings (token1, token2) unterschiedliche Metriken beziehungsweise Distanzen, sogenannte string comparison methods. Eine Darlegung und ein Vergleich der - Partial Ratiounterschiedlichen Methoden findet in Kapital 6 dieser Ausarbeitung statt. Im Rahmen der Implementation in Python existieren zusätzlich unterschiedliche Module, beispielweise „fuzzywuzzy“ oder „theFuzz“, die diesen Vergleich ermöglichen.

5.2 Anwendungsmöglichkeiten theFuzz

Abhängig von der Herangehensweise können mittels des Moduls theFuzz in Python unterschiedliche Vergleiche gezogen werden. Grundsätzlich verwendet theFuzz die Levenshtein-Distanz zur Berechnung der Unterschiede zwischen Sequenzen und ermöglicht die Auswahl

zwischen den Funktionen Simple Ratio, Partial Ratio, Token Sort Ratio, Token Set Ratio und extract.process. (pypi, n. d.)

Simple Ratio als Herangehensweise vergleicht beide Strings auf Basis aller Zeichen - die Reihenfolge der Zeichen ist hierbei relevant – und gibt als Ausgabe die Überschneidung beider Strings in Prozent an. Partial Ratio fungiert als Suchmethode ähnlich zu Simple Ratio, wodurch die Zeichenreihenfolge der zu vergleichenden Strings relevant ist. Es wird jedoch hierbei lediglich verglichen, ob String A in String B vorhanden ist. Sollte String B weitere Zeichen enthalten, wird trotzdem ein Ergebnis von 100 Prozent ausgegeben. Die Herangehensweise Token Sort Ratio vergleicht beide Input Strings auf Basis der einzelnen Token, wodurch die Reihenfolge der Token innerhalb der Strings nicht relevant ist. Token Set Ratio als Herangehensweise wirkt als Kombination der Methoden Partial Ratio und Token Sort Ration und vergleicht beide Input Strings auf Basis der Token, wobei String B weitere Zeichen enthalten kann. Extract.Process gilt als weiterführende Methode des Moduls und verwendet eine Kombination der genannten Methoden, um das beste Ergebnis zu erzielen.

Token Set Ratio

```
>>> fuzz.token_sort_ratio("fuzzy was a bear", "fuzzy fuzzy was a bear")
    84
>>> fuzz.token_set_ratio("fuzzy was a bear", "fuzzy fuzzy was a bear")
    100
```

Process

```
>>> choices = ["Atlanta Falcons", "New York Jets", "New York Giants", "Dallas Cowboys"]
>>> process.extract("new york jets", choices, limit=2)
    [('New York Jets', 100), ('New York Giants', 78)]
>>> process.extractOne("cowboys", choices)
    ("Dallas Cowboys", 90)
```

Abbildung 5.1: theFuzz Token Set Ratio und Process Funktion Quelle: pypi, n. d.

5.3 Auswahl geeigneter Kennzahlen der String Comparison Methods

Abhängig von den Anforderungen an die Suche sind verschiedene Methoden für den Vergleich zu wählen. Mögliche Metriken zur Messung der Distanzen zwischen unterschiedlichen Texten sind in Edit-based-similarities, Token-based-similarities und Sequence-based-similarities zu unterteilen.

Edit-based-similarities gehören zu den grundlegenden Methoden der Ähnlichkeitsbestimmung. Je mehr einzelne Operationen (Entfernen, Bearbeiten, Hinzufügen) durchzuführen sind, um eine Zeichenkette in eine andere umzuwandeln, desto größer ist der Abstand zwischen ihnen. Beispielsweise kann somit der Abstand zwischen den Namen „Thomas“ und „Tobias“ als 3 bestimmt werden. Da dieser Ansatz auf dem Vergleich einzelner Zeichen der Zeichenketten basiert, eignen sich Edit-based-similarities mehrheitlich für einzelne Wörter oder kurze Phrasen. Der Vergleich von Sätzen führt zu hohen Distanzen, wodurch das allgemeine Ergebnis schlechter ausfällt. Eine weitere Limitation dieser Vergleichsmethode ist, dass die semantische Bedeutung der Wörter nicht berücksichtigt wird, da lediglich einzelne Zeichen betrachtet werden (vgl. Cohen et al., 2003).

Token-based-similarities stellen eine komplexere Herangehensweise dar. Hier wird ein Text als Menge von Token – einzelnen Wörtern innerhalb der Zeichenkette – analysiert. Dies ermöglicht es, die semantische Bedeutung der Wörter zu berücksichtigen und längere Texte zu verarbeiten. Werden Methoden in Bezug auf Wortvektorrepräsentationen (Word2vec) gewählt, kann jedes Wort beschrieben und auf Basis des Kontexts verglichen werden. Auch die Verwendung eines Bag-of-Words-Ansatzes ermöglicht den Vergleich der semantischen Ähnlichkeit. Dies benötigt jedoch auch einen größeren Grunddatensatz, welche aus vollständigen Sätzen bestehen muss und führt zu einer komplexeren Berechnung, weshalb sich diese Methode weniger für den Vergleich kurzer Texteinheiten eignet (vgl. Augsten und Böhlen, 2014).

Sequence-based-similarities als Gruppe der Methoden zur Ähnlichkeitsbestimmung ähnelt letztendlich dem Ansatz der Edit-Distanzen, wobei kürzere Sequenzen innerhalb der Zeichenkette verglichen werden. Anstelle von einzelnen Zeichen werden Teilsequenzen der Wörter betrachtet. Die grundlegenden Vor- beziehungsweise Nachteile sind jedoch von den Edit-Distanzen übertragbar (vgl. Prasetya et al., 2018).

Distanzen wie die Levenshtein-Similarity oder Jaro-Winkler-Similarity sind den Methoden der Edit Distanzen zuzuordnen, wohingegen Cosine-Similarity eine Methode der Token based Distanzen und Longest-Common-Substring eine Methode der Sequence-based-Distanzen ist.

6 String Matching Methods

6.1 Levenshtein Distance

Die Levenshtein-Distanz ist ein Maß für die Differenz zwischen zwei Zeichenketten, welche 1965 von Wladimir Lewenstein vorgestellt wurde. Sie stellt die minimale Anzahl von Bearbeitungsoperationen (Einfügungen, Löschungen oder Ersetzungen) dar, die erforderlich sind, um eine Zeichenfolge in die andere umzuwandeln. Die Levenshtein-Distanz ist ein gängiges Maß für die Ähnlichkeit von Zeichenketten und wird in vielen Anwendungen wie der Rechtschreibprüfung, dem Fuzzy-Textabgleich und der DNA-Sequenzanalyse verwendet.

Die Berechnung der Levenshtein-Distanz erfolgt in der Regel mithilfe eines dynamischen Programmieransatzes. Für zwei Zeichenketten X und Y wird die Länge von X mit m und die Länge von Y mit n bezeichnet. Der Levenshtein-Abstand zwischen X und Y wird durch d(X, Y) dargestellt.

Der Abstand wird berechnet, indem eine Matrix der Größe (m+1)x(n+1) erstellt wird, wobei die Zelle an der Position (i, j) die minimale Anzahl von Editieroperationen darstellt, die erforderlich sind, um X[1...i] in Y[1...j] zu transformieren. Die Matrix wird von unten nach oben aufgefüllt, beginnend mit den Zellen (0, 0) bis (m, n).

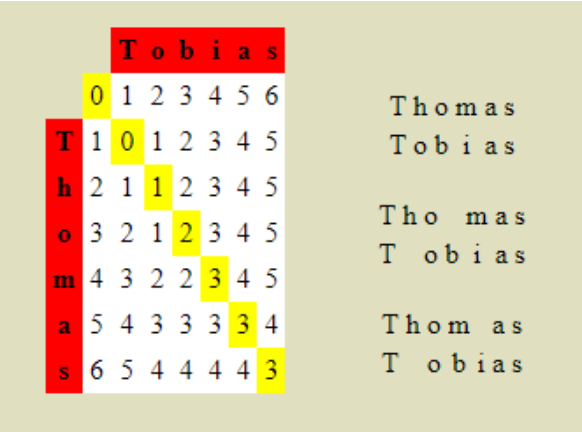

Abbildung 6.1: Levenshtein Matrix für exemplarischen Input. Darstellung nach „Levenshtein Simulator“, n. d.

Das Ergebnis von $d(m, n)$ stellt die minimale Anzahl an Editieroperationen dar, die notwendig sind um X in Y zu transformieren.

Formell lässt sich die Levenshtein Distanz wie folgt darstellen:

$$\operatorname{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \operatorname{lev}\left(\operatorname{tail}(a), \operatorname{tail}(b)\right) & \text{if } a[0] = b[0], \\ 1 + \min \begin{cases} \operatorname{lev}\left(\operatorname{tail}(a), b\right) \\ \operatorname{lev}\left(a, \operatorname{tail}(b)\right) \\ \operatorname{lev}\left(\operatorname{tail}(a), \operatorname{tail}(b)\right) \end{cases} & \text{otherwise,} \end{cases}$$

Abbildung 6.2: Formelle Beschreibung Levenshtein Distanz nach „Levenshtein Distance“, n. d.

Das Endergebnis von d(m, n) ist die minimale Anzahl von Editieroperationen, die erforderlich sind, um X in Y zu transformieren.

6.2 Jaro-Winkler Similarity

Das Jaro-Winkler-Ähnlichkeitsmaß ist ein String-Vergleichsalgorithmus, der die Ähnlichkeit zwischen zwei Strings berechnet. Er wird häufig bei der Datensatzverknüpfung und Entitätsauflösung verwendet, z. B. bei der Deduplizierung und dem Entitätsabgleich. Die grundlegende Jaro-Ähnlichkeit ist eine Zahl zwischen 0 und 1, wobei 0 für keine Ähnlichkeit und 1 für eine exakte Übereinstimmung steht. Die Jaro-Winkler-Ähnlichkeit wird berechnet, indem man die Jaro-Ähnlichkeit nimmt und einen Bonus für gemeinsame Präfixe hinzufügt, der die Ähnlichkeit für Zeichenketten erhöht, die ein gemeinsames Präfix haben. Die Gleichung für die Jaro-Winkler-Ähnlichkeit lautet:

$$Sim_w = Sim_j + lp(1 - sim_j)$$

Die Jaro Similarity ist das grundlegende Jaro Ähnlichkeitsmaß. l ist die Länge des gemeinsamen Präfixes am Anfang der Zeichenketten, bis zu einem Maximum von 4 Zeichen und p ist ein Skalierungsfaktor, der normalerweise auf 0,1 gesetzt wird. Die Jaro Ähnlichkeit wird auf der Grundlage der Anzahl der übereinstimmenden Zeichen in den beiden Zeichenketten und der Anzahl der Transpositionen (Umstellungen) von Zeichen berechnet, die erforderlich

sind, damit die Zeichenketten übereinstimmen. Die Grundformel für die Jaro Ähnlichkeit lautet:

$$Sim_j = 1/3(\frac{m}{|s_1|} + \frac{m}{|s_1 + 2|} + \frac{m_t}{|m|})$$

m ist die Anzahl der übereinstimmenden Zeichen in den beiden Zeichenfolgen. |s1| und |s2| sind die Längen der beiden Zeichenfolgen und t ist die Anzahl der Transpositionen, die erforderlich sind, damit die Zeichenfolgen übereinstimme. Der Ähnlichkeitsalgorithmus von Jaro berechnet zunächst die Anzahl der übereinstimmenden Zeichen in den beiden Zeichenfolgen, indem er jedes Zeichen in der ersten Zeichenfolge mit den Zeichen in einem bestimmten Fenster um dieselbe Position in der zweiten Zeichenfolge vergleicht. Die Größe des Fensters richtet sich nach der Länge der beiden Zeichenketten und ist definiert als das Maximum der Längen der beiden Zeichenketten geteilt durch 2, aufgerundet. Dann berechnet der Algorithmus die Anzahl der Transpositionen, die erforderlich sind, damit die Zeichenfolgen übereinstimmen, indem er die Positionen der übereinstimmenden Zeichen in den beiden Zeichenfolgen vergleicht und zählt, wie oft sich zwei übereinstimmende Zeichen nicht an derselben Position befinden.

Schließlich wird die Jaro Ähnlichkeit berechnet, indem der Durchschnitt der drei Werte genommen wird: das Verhältnis der übereinstimmenden Zeichen zur Länge der ersten Zeichenkette, das Verhältnis der übereinstimmenden Zeichen zur Länge der zweiten Zeichenkette und das Verhältnis der übereinstimmenden Zeichen minus der Anzahl der Transpositionen zur Anzahl der übereinstimmenden Zeichen.

Die Jaro Ähnlichkeit ist eine einfache und schnelle Methode, aber sie ist nicht sehr robust gegenüber Fehlern. Das Jaro-Winkler-Ähnlichkeitsmaß, das auch einen Bonus für gemeinsame Präfixe enthält, wird oft verwendet, um die Genauigkeit des Jaro-Ähnlichkeitsmaßes in Fällen zu verbessern, in denen die Zeichenketten wahrscheinlich ein gemeinsames Präfix haben. (vgl. Keil, 2019).

6.3 Longest Common Substring

Die Methode der längsten gemeinsamen Teilzeichenkette (Longest Common Substring, LCS) ist eine Möglichkeit, zwei Zeichenketten zu vergleichen und die längste gemeinsame zusammenhängende Teilzeichenkette zwischen ihnen zu finden. Die LCS-Methode wird

häufig in verschiedenen Anwendungen wie Textverarbeitung, Datenkompression und Sequenzanalyse eingesetzt.

Bei der LCS-Methode wird eine Matrix der Größe (m+1)x(n+1) erstellt, wobei m und n die Längen der beiden Zeichenketten X bzw. Y sind. Die Zelle an der Position (i, j) stellt die Länge der LCS von X[1...i] und Y[1...j] dar. Die Matrix wird von unten nach oben ausgefüllt, beginnend mit den Zellen (0, 0) bis (m, n).

Die Zellen werden nach den folgenden Regeln ausgefüllt:

$$L[i,0] = 0, \ \ for \ i = 0,1,2,...,m$$

$$L[0,j] = 0, \ \ for \ j = 0,1,2,...,n$$

$$L[i,j] = \{ \ L \ [i-1,j-1] + 1 if X[i] = Y[j] \ 0 if X[i] \neq Y[j]$$

Das Endergebnis von L[m, n] ist die Länge der LCS von X und Y. Sobald die Matrix ausgefüllt ist, kann man von der Zelle (m, n) zurückverfolgen, um die tatsächliche LCS zu finden, indem man den Weg der Zellen mit den Maximalwerten verfolgt. Dies kann geschehen, indem man die Eltern jeder Zelle verfolgt und dem Pfad der Eltern mit dem größten Wert folgt.

Die LCS-Methode ist ein schneller und effizienter Weg, um zwei Zeichenketten zu vergleichen und die längste gemeinsame zusammenhängende Teilzeichenkette zwischen ihnen zu finden. Die Methode findet in verschiedenen Bereichen Anwendung, darunter Textverarbeitung, Datenkompression und Sequenzanalyse.

Allerdings ist die Aussagekraft von absoluten Werten bei der LCS vorsichtig zu interpretieren. Um die Aussagekraft zu erhöhen, gibt es Erweiterungen der klassischen LCS Methode, bei denen z.B. noch die Länge der Inputs berücksichtigt wird.

7 Zusammenfassung

7.1 Fazit

NER kann bei Finetuning durch einen ausbalancierten, globalen Datensatz, eine wertvolle Technik sein, um Geschäftspartner in einem Dokument. Im Vergleich zu Word Embedding-Methoden bietet NER eine präzisere Möglichkeit, Geschäftspartner zu erkennen, indem es direkt nach spezifischen Entitäten wie Namen, Orten und Organisationen sucht. Mit einem Framework wie Streamlit kann das Ergebnis von NER visuell dargestellt werden, was es Benutzern ermöglicht, die Daten schnell und intuitiv zu überprüfen. Darüber hinaus kann die Fuzzy-Suche innerhalb von NER verwendet werden, um ähnliche oder fast identische Namen zu erkennen, was die Genauigkeit der Identifizierung von Geschäftspartnern weiter verbessert. Insgesamt kann gesagt werden, dass NER eine praktische und effektive Methode ist, um Geschäftspartner in großen Datensätzen zu identifizieren und zu visualisieren. Es ist jedoch wichtig, dass die Methode regelmäßig überprüft wird, um sicherzustellen, dass die Ergebnisse korrekt und aktuell sind. Allgemein wurde erforscht, dass die Kombinationsmethode theFuzz, welche auf der Levenshtein Distanz basiert, im Kontext der fuzzy Suche auf einem PDF am besten funktioniert und die Aussagekräftigsten ist.

7.2 Ausblick

Der Ausblick für die Anwendung von NER zur Identifizierung von Geschäftspartnern ist sehr positiv. In einer immer digitaleren Welt sammeln Unternehmen und Organisationen immer mehr unstrukturierte Daten, und NER ist eine wertvolle Technik, um diese Daten zu verarbeiten und zu analysieren. In Zukunft wird es wahrscheinlich weitere Fortschritte in der Technologie geben, die es ermöglichen, NER noch präziser und effizienter zu gestalten. Dazu könnten neue Algorithmen und maschinelles Lernen beitragen, die eine höhere Genauigkeit und eine bessere Verarbeitung von unstrukturierten Daten ermöglichen. Darüber hinaus werden wir wahrscheinlich eine Integration von NER in andere Systeme und Workflows sehen, was es Benutzern ermöglicht, die Daten noch schneller und einfacher zu verarbeiten.

Literaturverzeichnis

Almeida, F., & Xexéo, G. (2019). Word embeddings: A survey. *arXiv preprint arXiv:1901.09069*.

Augsten, N., & Böhlen, M. H. (2014). Token-Based Distances. In *Similarity Joins in Relational Database Systems* (S. 25–59). Springer.

base_bert_NER [Accessed: 2023-01-30]. (n. d.).

Church, K. W. (2017). Word2Vec. *Natural Language Engineering*, *23*(1), 155–162.

Cohen, W. W., Ravikumar, P., Fienberg, S. E., et al. (2003). A Comparison of String Distance Metrics for Name-Matching Tasks. *IIWeb*, *3*, 73–78.

Das, N., Ghosh, S., Gonçalves, T., & Quaresma, P. (2014). Comparison of different graph distance metrics for semantic text based classification. *Polibits*, (49), 51–58.

EuropeanCommission. (2020). *Consolidated list of persons, groups and entities subject to EU financial sanctions*. https://data.europa.eu/data/datasets/consolidated-list-of-persons-groups-and-entities-subject-to-eu-financial-sanctions?locale=en

Fu, Z., Wu, X., Guan, C., Sun, X., & Ren, K. (2016). Toward efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement. *IEEE Transactions on Information Forensics and Security*, *11*(12), 2706–2716.

Keil, J. M. (2019). Efficient bounded Jaro-Winkler similarity based search. *BTW 2019*.

Lai, L.-F., Wu, C.-C., Lin, P.-Y., & Huang, L.-T. (2011). Developing a fuzzy search engine based on fuzzy ontology and semantic search. *2011 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2011)*, 2684–2689.

Levenshtein Distance [Accessed: 2023-01-30]. (n. d.).

Levenshtein Simulator [Accessed: 2023-01-30]. (n. d.).

Li, B., & Han, L. (2013). Distance weighted cosine similarity measure for text classification. *Intelligent Data Engineering and Automated Learning–IDEAL 2013: 14th International Conference, IDEAL 2013, Hefei, China, October 20-23, 2013. Proceedings 14*, 611–618.

Masek, W. J., & Paterson, M. S. (1980). A faster algorithm computing string edit distances. *Journal of Computer and System sciences*, *20*(1), 18–31.

Prasetya, D. D., Wibawa, A. P., & Hirashima, T. (2018). The performance of text similarity algorithms. *International Journal of Advances in Intelligent Informatics*, *4*(1), 63–69.

pypi. (n. d.). *thefuzz 0.19.0*. https://pypi.org/project/thefuzz/

Rahutomo, F., Kitasuka, T., & Aritsugi, M. (2012). Semantic Cosine Similarity.

Ukkonen, E. (1983). On approximate string matching. *Foundations of Computation Theory: Proceedings of the 1983 International FCT-Conference Borgholm, Sweden, August 21–27, 1983 4*, 487–495.

Wang, B., Yu, S., Lou, W., & Hou, Y. T. (2014). Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud. *IEEE INFOCOM 2014-IEEE conference on computer communications*, 2112–2120.