Universität Leipzig

Seminar: Digitalisierung und Analyse von Pop-Lyrics

Fakultät für Mathematik und Informatik

Kurs: Aktuelle Trends in den Digital Humanities

Dozent: Roman Schneider

WS 2023/24

Eigennamenerkennung in englischsprachigen Songtexten

Ein analytischer Vergleich der Python-Packages spaCy, Flair und NLTK

Name: Rücker Florian

Anschrift: Scheffelstraße 48C, Leipzig

Email: fr87gydy@studserv.uni-leipzig.de

Matrikelnummer: 3728417

Abgabe: 31.03.2024

Inhaltsverzeichnis

1. Einleitung

Die Eigennamenerkennung (Named Entity Recognition, NER) ist ein wichtiger Aspekt des Natural Language Processing (NLP). Sie spielt eine bedeutende Rolle in vielen Anwendungen wie der Informationsextraktion, maschinellen Übersetzen, Frage-Antwort-Systemen (Chatbot Modellen) und Textzusammenfassung. Das Ziel der Eigennamenerkennung ist spezifische Entitäten wie Personen, Orte, Organisationen und andere benannte Objekte in einem Text zu identifizieren und zu klassifizieren. Somit werden Informationen genauer erfasst und die automatisierte Analyse von Texten erleichtert. Trotz der offensichtlichen Fortschritte, wie es an Beispielen von modernen K.I. Modellen, wie ChatGPT zu sehen ist, gibt es dennoch Schwierigkeiten in der NER. Vor allem die NER in unstrukturierten und kreativen Texten, wie sie in Songtexten zu finden sind, bleiben eine Herausforderung. Diese Hausarbeit konzentriert sich auf die Anwendung und den Vergleich von den drei weit verbreiteten Python-Packages Flair, NLTK und spaCy zur Eigennamenerkennung in englischsprachigen Songtexten. Songtexte stellen eine besondere Herausforderung für die NER dar, da sie oft metaphorisch sind und eine flexible Grammatik verwenden. Diese können im von strengeren grammatikalischen Regeln abweichen, welche im Vergleich in formelleren Textarten zu finden sind. Darüber hinaus können Songtexte eine Vielzahl von kulturellen Referenzen und Slang Ausdrücke enthalten, die für NER-Modelle schwer zu identifizieren sind und die Klassifizierung von Eigennamen erschwert. Ziel dieser Hausarbeit ist es, die Leistung der drei Python-Packages zu vergleichen und zu analysieren. Anhand des Vergleichs auf denselben Datensatz können wir die Stärken und Einschränkungen der Packages besser extrahieren. Dabei ist es natürlich auch interessant herauszufinden, welches der Packages am besten für die Anwendung in diesen spezifischen Kontext geeignet ist. Dafür werden verschiedene Aspekte wie die Genauigkeit, die Fehlerrate, die Überschätzungsquote und die Fähigkeit korrekte Kategorien zuzuweisen berücksichtigt.

2. Eine Einführung in die Eigennamenerkennung

Die Eigennamenerkennung (Named Entity Recognition, NER) stellt einen wichtigen Schritt im Bereich des Natural Language Processing (NLP) dar, der sich mit der Identifizierung und Klassifizierung spezifischer Entitäten in einem Text befasst. (Biemann 2022, S.106) Sie wird in der Regel mit Sequenztagging-Ansätzen implementiert, die neben annotiertem Trainingstext, auch auf Listen bekannter Namen und Namensteile zurückgreifen können.

(Biemann 2022, S.106) Die NER-Systeme verwenden oft Gazetteers, Listen oder Verzeichnisse von benannten Entitäten wie Personen, Orten, Organisationen und anderen spezifischen Objekten oder Begriffen, um die Entitäten im Text zu identifizieren. (Biemann 2022, S.106) Diese Gazetteers sind eine wichtige Ressource bei der Unterstützung der NER-Verfahren, obwohl sie alleine nicht ausreichen, denn sie sind sowohl unvollständig als auch anfällig für Mehrdeutigkeiten. (Biemann 2022, S.106) Aus diesem Grund ist es wichtig, dass die NER-Modelle auch auf annotierte Trainingsdaten zurückgreifen, um die Leistungsfähigkeit und Genauigkeit zu verbessern. Die Disambiguierung von Eigennamen ist ein wichtiger Schritt innerhalb der NER. Es soll eine eindeutige Zuordnung von einer spezifischen Entität aus einem Text erfolgen, wenn der Name dieser Entität mehrdeutig ist oder auf mehrere möglichen Entitäten verweisen kann. Dieser Prozess ist wichtig, um sicherzustellen zu können, dass die identifizierten Entitäten korrekt und eindeutig zugeordnet werden. (Biemann 2022, S.106) In den meisten NER-Tagsets werden vier Klassen unterschieden: Personennamen (PER), Ortsnamen (LOC), Organisationsnamen (ORG) und Sonstige (MISC). (Biemann 2022, S.106) Die NER-Systeme verwenden diese Tagsets, um die erkannten Entitäten entsprechend zu klassifizieren und zu kennzeichnen. (Biemann 2022, S.106) Die Klassifizierung in verschiedene Kategorien ermöglicht eine präzise Identifizierung und Unterscheidung der verschiedenen Arten von Entitäten in einem Text.

3. Erstellung des Datensatzes

Den Datensatz habe ich selbst, mithilfe der Songtexte von der Webseite „Genius‘‘(Genius, 2024), erstellt. Genius bietet sich für meinen Verwendungszweck besonders gut an, da Passagen aus Liedtexten und auch einfache Wörter im Kontext des Songs erklärt werden. Mit dieser Hilfestellung kann ich eigenständig überprüfen, ob es sich bei einem Wort, oder einer Wortfolge, um einen Eigennamen, oder einer metaphorischen Variation handelt. Bei der Erstellung des Datensatzes habe ich Wert daraufgelegt, dass in meinem englischsprachigen Songtextkorpus alle vier vorher beschriebenen NER-Tagsets vertreten sind. Also (LOC), (PER), (ORG) und (MISC). Ohne größeres Wissen über eine große Bandbreite von Songs gestaltete sich die Suche nach den NER-Tagsets (ORG) und (MISC) als schwierig. Die Suche nach den NER-Tagsets (PER) und (LOC) gestaltete sich deutlich einfacher. Wobei das sehr gut an der Repräsentierbarkeit der NER-Tags in den gesamt existierenden Songs liegen kann.

So waren in Amerika von 1960s-2000s im Durchschnitt 67,3% der Lieder Liebeslieder. (Christenson et al. 2018, S.7) Eine Korrelation zur häufigen Verwendung von (PER) ist somit gegeben. Anders gestaltete sich dies bei (LOC) und (ORG). Repräsentative Songs für beide NER-Tagsets sind einfach zu finden. Wobei für den Zweck der häufigeren Frequenz von Eigennamen in einem Song der NER-Tagset (LOC) einfacher zu finden ist. In Songs über Reisen kommen häufig viele Städtenamen, oder Ländernamen vor. Anders bei (ORG), denn dort konzentriert sich der Songtext häufig auf nur einen (ORG) und der NER-Tag (ORG) steht nach meiner Recherche meistens schon im Songtitel. Zudem besteht die Möglichkeit von gesetzten Produkt Platzierungen in den Songtexten, die für die Künstler eine zusätzliche Einnahmequelle darstellen können. (Craig et al., S.1) Aus diesem Grund ist die Repräsentierbarkeit von (ORG) in meinem Datensatz unterrepräsentativ im Vergleich zu den NER-Tagsets (LOC) und (PER). Das Selbe Problem bestand auch bei dem NER-Tagset (MISC). Hier muss sich der Song nicht zwangsläufig auf den NER-Tagset beziehen, allerdings ist es schwierig ihn zu finden, wenn man wie bei den NER-Tagset (ORG) nicht genau nach den Eigennamen sucht. Das sind die Erfahrungen, die ich bei der Erstellung des Songtextkorpus gemacht habe. Für die Textnormalisierung der Songtexte habe ich zuerst versucht alle Satzzeichen zu entfernen. Dies stellte sich aber bei allen Packages als Fehler heraus. Wenn zwei Eigennamen zusammenstehen, werden sie nicht als zwei getrennte Eigennamen betrachtet, sondern diese zwei Eigennamen werden zu einem Eigennamen zusammengefasst. Am Beispiel von zwei Nachnamen würde also aus (Nixon), (Kennedy) zu (Nixon Kennedy) werden. Da einige Eigennamen von vorneherein nicht durch Satzzeichen getrennt wurden, was durch die Versgestaltung in einigen Songs häufig gegeben war, habe ich diese falls nötig am Ende des Verses hinzugefügt. Damit habe ich die eine einzelne Erfassung aller Eigennamen garantiert. Danach habe ich mit Hilfe von NLTK alle Stoppwörter entfernt. Doch auch dies führte nicht zu einer Verbesserung meines Datensatzes. Ich habe die Auswertung der erkannten Eigennamen, nachdem die Stoppwörter entfernt wurden, mit der Auswertung des Datensatzes verglichen, in dem die Stoppwörter nicht entfernt wurden. Zu meiner Überraschung stellte ich fest, dass die Methode, welche ich über NLTK „from nltk.corpus import stopwords“ verwendete, einige der ausgewerteten Eigennamen entfernte. Somit war die Genauigkeit mit dem Entfernen der Stoppwörter schlechter. Ohne mir erklärbaren Grund wurden Stadtnamen entfernt, welche meiner Ansicht nach keine Stoppwörter waren. Zum Beispiel erkannte die Eigennamenerkennung von spaCy in dem Liedtext „London Calling‘‘ von „The Clash‘‘ (Genius 2024), anstelle von („GPE‘‘, „London‘‘, 12) nun nur noch („GPE‘‘, „London‘‘, 9). Warum das passiert ist, ist mir

unbekannt. Anschließend versuchte ich mit spaCy Stoppwörter zu entfernen und den Songtext zu lemmatisieren. Dies funktionierte bei allen drei Packages, ohne den Verlust von Eigennamen. Allerdings half es auch nicht die Genauigkeit zu verbessern, sondern blieb gleich. Das wird wahrscheinlich darauf zurückzuführen sein, dass die Lemmatisierung in der Regel nicht sinnvoll für Eigennamen ist. Denn diese haben in der Regel keine morphologischen Variationen, die auf einen gemeinsamen Wortstamm zurückgeführt werden könnten. Eigennamen behalten in der Regel ihre Form bei, unabhängig vom Kontext, in dem sie verwendet werden. Aus dem Grund habe ich beschlossen auch die Lemmatisierung nicht auf den Songtext zu verwenden, um Komplikationen wie mit den Stoppwörtern über die NLTK-Methode zu verhindern. Für Überprüfungszwecke ist die Lemmatisierungsmethode Exemplarisch im spaCy Modell enthalten. Abschließend habe ich selbst einen Korrekturdatensatz erstellt. Im Korrekturdatensatz habe ich selbständig annotiert, zu welchem Eigennamen welcher NER-Tagset gehört, den Eigennamen selbst und die Frequenz, wie oft der Eigenname insgesamt im Songtextkorpus vorkommt („NER-Tag“, „Eigenname“, Frequenz). Dies zählt alle Eigennamen auf und dient als Kontrolle zu den Ergebnissen der Packages NLTK, Flair und spaCy.

4. Methodik

4.1 Vergleich von spaCy, Flair und NLTK für die Eigennamenerkennung

In dieser Untersuchung werde ich die drei weit verbreiteten Natural Language Processing (NLP)-Bibliotheken, Flair, spaCy und NLTK, hinsichtlich ihrer Fähigkeiten zur Eigennamenerkennung vergleichen.
Flair ist eine NLP-Bibliothek, die von der Humboldt-Universität Berlin entwickelt wurde. (Flair 2024) Sie ermöglicht es, modernste NLP-Modelle auf Ihren Text anzuwenden, wie z.B. die Eigennamenerkennung (NER), Sentiment Analyse und Part-of-Speech-Tagging (PoS). (Flair 2024) Zudem bietet Flair Schnittstellen, die es ermöglichen, verschiedene Wort- und Dokumenteinbettungen zu verwenden und zu kombinieren. (Flair 2024) Dabei bietet Flair eine integrierte Eigennamenerkennungsfunktion namens SequenceTagger, die auf maschinellem Lernen basiert und Eigennamen sowie andere benannte Entitäten in Texten identifizieren kann. (Harsh 2020, S.1) SpaCy hingegen ist eine etablierte Open-Source-NLP-Bibliothek, die speziell für den industriellen Einsatz entwickelt wurde. (SpaCy 2024)
Sie bietet eine breite Palette von Funktionen, darunter auch die Eigennamenerkennung.

(Spacy 2024) SpaCy verwendet vortrainierte statistische Modelle, die auf großen Textkorpora trainiert wurden, um Eigennamen zu identifizieren. Diese Modelle basieren auf maschinellem Lernen und einer neuronaler Netzwerkarchitektur. (SpaCy 2024)
NLTK (Natural Language Toolkit) ist eine Python-Bibliothek und ist für die Verarbeitung von menschlichen Sprachdaten entwickelt worden. (NLTK 2024) NLTK bietet Werkzeuge und Ressourcen für die Verarbeitung natürlicher Sprache, darunter auch die Eigennamenerkennung. (NLTK 2024) Im Gegensatz zu Flair und spaCy, die auf modernen maschinellen Lernansätzen basieren, verwendet NLTK traditionellere statistische Modelle und Regelwerke für die Entitäten Erkennung, wie z.B. Chunking und Named Entity Recognition (NER) auf der Grundlage von Mustern und Regeln. (Bird et al. 2009, S. 284)
Alle Python Codes, die im Folgenden Abschnitt verwendet werden, wurden auf der Python Version = 3.7.10 ausgeführt und sind im Literaturverzeichnis zu finden. Außerdem sind alle Ergebnisse mit dem Ausführen des Codes einsehbar.

4.1.1 NER mit SpaCy

Zuerst habe ich alle notwendigen Bibliotheken heruntergeladen und importiert. SpaCy ist eine Bibliothek für fortgeschrittene Natural Language Processing in Python. ,,defaultdict‘‘ ist ein spezieller Typ für einen Dictionary aus der collections-Bibliothek, welcher einen Standardwert für nicht existierende Schlüssel bereitstellt. Anschließend habe ich STOP_WORDS importiert. Dies ist eine Liste von häufig verwendeten Wörtern (wie “the”, “is”, “and”, etc.) im englischen die oft aus dem Text entfernt werden. Das wird in der Textbereinigung eingesetzt, bevor NLP-Aufgaben durchgeführt werden. Anschließend habe ich das spaCy- Modell geladen. Hierbei gab es verschiedene Möglichkeiten. Es ist möglich drei verschieden Modelle zu laden: ,,en_core_web_sm, en_core_web_md und en_core_web_lg‘‘. Der Unterschied ist ausschließlich die Größe der Modelle nach der Reihenfolge small, medium und large. Diese Modelle sind in der Lage verschiedene NLP-Aufgaben im englischen durchzuführen. Zudem Tokenisierung, Part-of-Speech-Tagging und Named Entity Recognition, welches für meine Arbeit wichtig ist. Ich bin erst einmal davon ausgegangen, dass das Training auf dem größeren Modell am sinnvollsten wäre, da dies ein höheres Repertoire an Eigennamen haben sollte. Die Anwendung der Modelle medium und large zeigte folgendes Ergebnis. Die Genauigkeit der gefundenen Eigennamen verglichen mit meinen Korrekturdatensatz war bei large höher, ~68,81% zu ~63,36% beim medium Modell. Allerdings hat das Größere Modell mehr Eigennamen gefunden, als tatsächlich im

Songtextkorpus existent waren, was zu einer Überschätzung führte. Die Überschätzung lag beim medium Modell bei ~1,98% und beim large Modell bei ~7,92%. Ich habe es als wichtiger bewertet, dass das Modell sich nicht überschätzt und daher das Modell medium weiterverwendet.
Nachdem ich die Wahl meines Modelles abgeschlossen habe, wird anschließend der Text aus dem Songkorpus eingelesen und in der Variable ,,songs‘‘ gespeichert. Dann habe ich das Model angewandt. Das Modell gibt dann ein Doc-Objekt zurück, das eine Sequenz von Token enthält, auf die verschiedene Attribute und Methoden angewendet werden können.
Anschließend habe ich den Text lemmatisieren lassen und die Stoppwörter entfernt. Danach wird für jeden Token im Doc-Objekt wird überprüft, ob das Lemma des Tokens (die Grundform des Wortes) ein Stoppwort ist. Wenn nicht, wird das Lemma des Tokens zur Liste ,,lemmatized_text‘‘ hinzugefügt.
Nun muss ein Dictionary erstellt werden, welches zur Speicherung der Häufigkeit jeder Entity in jeder Kategorie dient. Dabei wird ein ,,defaultdict‘‘ namens ,,entity_freq‘‘ erstellt, um die Häufigkeit jeder erkannten Entity in jeder Kategorie zu speichern.
Für jede Entity, die vom spaCy-Modell erkannt wurde, wird überprüft, ob die Kategorie der Entity eine der folgenden ist: ‘PERSON’, ‘ORG’, ‘LOC’, ‘GPE’ und ‘MISC’. Diese Kategorien habe ich aus zwei Gründen ausgewählt. Zum einen beschränkt sich diese Arbeit und die Erklärung von NER auf : ‘PERSON’, ‘ORG’, ‘LOC’ und ‘MISC’. ‘GPE’ ist eine weitere Kategorie, die sich wie ‘LOC’ auf Orts, oder Ländernamen bezieht. Doch wird dies in SpaCy mit ‘GPE’ genauer kategorisiert und das Auslassen von ‘GPE’ hätte zur Folge, dass weniger Ortsnamen erkannt werden. Der zweite Grund ist, dass die anderen Modelle, wie Flair und NLTK nicht alle NER-Kategorien erkennen können wie SpaCy. Produktnamen zum Beispiel könnte spaCy erkennen, aber alle anderen Modelle nicht. So könnte es zu keinem Vergleich kommen und es wurden hier nur die üblichsten und übereinstimmenden NER Kategorien gewählt.
Wenn nun das spaCy Modell eine NER-Kategorie erkennt, wird die Häufigkeit dieser Entität in ihrer Kategorie im entity_freq-Dictionary erhöht.
Dann habe ich meinen Korrekturdatensatz eingefügt, der die korrekten Häufigkeiten für alle Entitäten enthält. Dieser wird später verwendet, um die Genauigkeit des Modells zu berechnen. Anschließend habe ich eine Methode geschrieben, die die Gesamtzahl der korrekten Entitäten berechnet, indem die Häufigkeiten aller Entitäten im Korrekturdatensatz summiert werden. Für einen Vergleich wird nun die Gesamtzahl der vom Modell erkannten

Entitäten berechnet, indem die Häufigkeiten aller Entitäten im entity_freq-Dictionary summiert werden.
Ich will nun herausfinden, wie viele Entitäten das Modell erkannt hat, die nicht Teil des Korrekturdatensatzes sind. Also wo hat sich das Modell überschätzt. Der Prozentsatz der falschen Entitäten wird berechnet, indem die Anzahl der erkannten Entitäten durch die Anzahl der korrekten Entitäten geteilt und mit 100 multipliziert wird.
Die Überschätzungsquote lag bei ~1,98%. Für einen genaueren Einblick lasse ich mir über die Methode ,,def extra_Entitäten(correction_data, entity_freq):‘‘ alle Entitäten ausgeben, die das Modell zusätzlich gefunden hat. Anschließend wird die Differenz zwischen dem Korrekturdatensatz und der entity_freq-Dictionary berechnet. Also alle vom Modell gefundenen Entitäten, welche auch im Korrekturdatensatz enthalten sind. Dies wird gemacht, indem für jede Entität im Korrekturdatensatz der Unterschied zwischen der im entity_freq-Dictionary gefundenen Häufigkeit und der korrekten Häufigkeit summiert wird. Mit der Methode ,,def missing_Entitäten(correction_data, entity_freq):‘‘ habe ich mir ausgeben lassen, wie viele Entitäten aus dem Korrekturdatensatz nicht gefunden wurden. Wenn eine Entität erkannt wurde, aber die Kategorie falsch ist, wird der Fehler als 0,5 anstelle von 1 gezählt. Dies habe ich eingeführt, um die tatsächliche Genauigkeit zu erhöhen. Zwar wurde die NER-Kategorie nicht erkannt, aber dass es sich um einen Eigennamen handelte. Der Prozentsatzfehler wird berechnet, indem die Gesamtfehleranzahl durch die Gesamtzahl der korrekten Entitäten geteilt und mit 100 multipliziert wird. Letzten Endes wird die Genauigkeit des Modells berechnet und ausgegeben, indem der Prozentsatzfehler von 100 subtrahiert wird. Die Genauigkeit betrug dann ~63,37%.

4.1.2. NER mit Flair

Ich habe erneut alle benötigten Module importiert, einschließlich Sentence und SequenceTagger aus Flair, sowie defaultdict aus der Python-Standardbibliothek, um ein leeres Wörterbuch mit Standardeinträgen zu erstellen. Als nächstes wird das vortrainierte NER-Modell (Named Entity Recognition) aus Flair geladen. Dieses Modell wurde bereits auf großen Datensätzen trainiert, um Eigennamen in Texten zu identifizieren. Anschließend wird der Text aus dem Songkorpus gelesen und in der Variable songs gespeichert. Darauffolgend muss ein Sentence-Objekt erstellt werden, welches den gelesenen Text enthält. Dann wird das NER-Modell auf das Sentence-Object angewendet, um die Entitäten im Text zu erkennen. Mit defaultdic wird wieder ein leeres Wörterbuch entity_freq erstellt, um die Häufigkeit der

erkannten Entitäten zu speichern. Dabei werden erneut die Entitäten 'MISC', 'LOC', 'PER' und 'ORG' durchlaufen und die Häufigkeit gezählt. Anpassungen mussten hierbei bei den Namen der Entitätskategorien gemacht werden. Ortsbezogene Eigennamen werden ausschließlich unter 'LOC' erkannt und aus 'PERSON' wird 'PER'. Anschließend wird wieder die Überschätzungsquote berechnet. Die Funktionen dafür sind genau dieselben, wie ich sie auch bei spaCy verwendet habe. Die Überschätzungsquote liegt hier bei: ~13,86%. Anschließend wird die Differenz zwischen dem Korrekturdatensatz und der entity_freq-Dictionary berechnet. Wenn eine Entität erkannt wurde, aber die Kategorie falsch ist, wird der Fehler als 0,5 anstelle von 1 gezählt. Nach der Berechnung des Prozentsatzfehlers betrug die Genauigkeit dann: ~63.96%.

4.1.3 NER mit NLTK

Erneut werden die benötigten Module und Funktionen importiert,
darunter nltk, defaultdict aus dem collections-Modul, und verschiedene Funktionen
aus nltk wie ne_chunk, pos_tag und word_tokenize.
Danach habe ich die Methode get_continuous_chunks(text) definiert. Zuerst wird der Text mit der Funktion word_tokenize(text) in Wörter zerlegt. Dieser Schritt tokenisiert den Text und wandelt den Text in eine Liste von Wörtern um. Anschließend wird mit der Funktion pos_tag(...). jedem Wort ein Part-of-Speech-Tag. Dieser Schritt kennzeichnet jedes Wort mit seinem grammatikalischen Typ (z.B. Nomen, Verb, Adjektiv usw.) aus.
Danach wird mit der Funktion ne_chunk(...) eine NER durchgeführt. Dieser Schritt erkennt Named Entitäten im Text und gruppiert sie in “Bäume”, wobei jeder Baum eine Entität darstellt. Diese die Funktion durchläuft jeden “Baum” in den “chunked” Daten. Wenn der aktuelle Eintrag ein Baum ist (was bedeutet, dass es sich um eine Named Entity handelt), fügt die Funktion die Named Entity und ihren Typ zum aktuellen “Chunk” hinzu. Wenn der aktuelle Eintrag kein Baum ist und es einen aktuellen “Chunk” gibt, fügt die Funktion den aktuellen “Chunk” zur Liste der “continuous_chunk” hinzu und leert den aktuellen “Chunk”. Wenn der aktuelle Eintrag kein Baum ist und es keinen aktuellen “Chunk” gibt, fährt die Funktion fort. Letzten Endes gibt die Funktion die Liste continuous_chunk zurück, die alle erkannten Named Entitäten und ihre Typen enthält. Dann wird der Text aus dem Songkorpus gelesen und in der Variable songs gespeichert. Nun wird die Funktion get_continuous_chunks auf die songs angewendet, um die Named Entitäten zu extrahieren. Dabei werden erneut die

Entitäten 'MISC', 'LOC', 'PER' und 'ORG' durchlaufen und die Häufigkeit gezählt. Anpassungen mussten auch hierbei bei den Namen der Entitätskategorien gemacht werden. 'ORG' wird in diesem Modell nur als 'ORGANIZATION' erkannt, 'PER' als 'PERSON' und neben 'LOC' existiert auch wieder 'GPE'.
Anschließend wird wieder die Überschätzungsquote berechnet. Die Funktionen dafür sind genau dieselben, wie ich sie auch bei spaCy verwendet habe. Die Überschätzungsquote liegt hier bei: ~23,76%. Anschließend wird die Differenz zwischen dem Korrekturdatensatz und dem entity_freq-Dictionary berechnet. Wenn eine Entität erkannt wurde, aber die Kategorie falsch ist, wird der Fehler als 0,5 anstelle von 1 gezählt. Nach der Berechnung des Prozentsatzfehlers betrug die Genauigkeit dann: ~52,47%.

4.2. Auswertung und Bewertung aller Modelle

Ich habe alle drei Modelle getestet, deren Genauigkeit im Vergleich zum Korrekturdatensatz berechnet, sowie deren Überschätzungsquote berechnet. Für die Auswertung und genauere Betrachtung aller Modelle habe ich zusätzlich einen Code geschrieben, der mir alle von den Modellen gefundenen Eigennamen ausgibt. Dabei hatte das NLTK-Modell am meisten Schwierigkeiten. Zum einen hatte es eine Überschätzungsquote von ~26,9%. Was bedeutet, dass es viele falsche positive Entitäten erkannte. Ein Beispiel dafür ist (GPE, Engines, 2), (GPE, No, 4) oder (GPE, Bright, 1). Keine Ortsnamen, aber mitunter aufgrund der Großschreibung am Anfang jedes Verses als solche erkannt. Das erklärt die hohe Überschätzquote. Eine songs.lower() Methode kommt nicht in Frage, weil alle Eigennamen großgeschrieben werden. Eine Methode zur songs.lower() für alle Wörter mit Ausnahme von Nomen ist auch nicht empfehlenswert. Wenn ein fiktiver Charakter den Namen „Small Man“ besitzt, ein Eigenname also, würde dieser zu „small Man“ werden. In dem Zusammenhang würde nur „Man“ als Eigenname erkannt werden. Das beschriebene Problem stellt die Herausforderung von Songtexten dar. Die Genauigkeit im Vergleich zum Korrekturdatensatz betrug 53,3%. Erklärt werden kann dies zum einen dadurch, dass es auch falsche Zuweisungen von Entitäten, wie (ORGANIZATION, Kilimanjaro, 1) anstelle von (LOC, Kilimanjaro, 1) gab, welche die Genauigkeit minderten. Bemerkenswert war auch, dass NLTK keine MISC oder LOC erkannte, obwohl diese im Korrekturdatensatz vorhanden waren. Das Modell hatte auch sehr große Schwierigkeiten Personen zu erkennen. Woran genau das liegt, kann ich nicht rückführend beantworten. Sowohl nur Nachnamen ('PERSON', 'Eisenhower'), sowie Vor- und Nachname ('PERSON', 'Euell Gibbons') wurden nicht erkannt.

Ich habe berechnet, wie viele Personen nicht erkannt wurden, im Vergleich zu allen Personen im Korrekturdatensatz. Insgesamt wurden ~21,9% Personen nicht erkannt. Das ist sehr viel und wie bereits ausgeführt nur ein Faktor von den vorherigen. Dies erklärt jedoch auch die geringe Genauigkeit.

Das Flair-Modell hatte eine Überschätzungsquote von ~13,86%. Das ist immer noch hoch, auch wenn es weniger als das NLTK-Modell beträgt und mit dem Hinblick darauf, dass idealerweise nahe 0% angestrebt wird. Erneut haben wir das Problem, dass einige Wörter als Eigennamen aufgrund der Großschreibung erkannt werden (PER, Gonna, 4) und (MISC, See, 1) zum Beispiel. Doch sind es weniger als beim NLTK-Modell. Ein weiteres Problem war, dass häufig mehrere Wörter, als ein zusammenhängender Eigenname betrachtet wurde („Oh Lord, won't you buy me a Mercedes Benz?") oder ('Stranger in a Strange Land'). Dabei war häufig Ausschlag gebend, dass ein Vers großgeschrieben anfing und in dem Vers ein Eigenname enthalten war. Die Genauigkeit im Vergleich zum Korrekturdatensatz betrug ~63,61%. Also ungefähr 10% besser als das vorherige Modell. Flair erkennt „LOC“ und „MISC“ und teilt diesen auch Eigennamen zu. Es hatte keine Probleme Personennamen zu identifizieren, die einen Vor- und Nachnamen hatten. Jedoch scheiterte das Modell ab und zu bei nur Einzelnamen, wie ('PER', 'Eisenhower') und ('PER', 'Dylan'). Einige LOC wurden auch nicht als solche entdeckt, wie ('LOC', 'The River Kwai') und ('LOC', 'Reno'). Zudem wurden auch bis auf zwei keine ORG-Eigennamen entdeckt.

Das spaCy Modell hatte die beste Überschätzungsquote mit nur ~1,98%. Die Probleme die spaCy hatte, waren zu einem großen Teil auf die Natur von Songtexten zu finden und das ist die metaphorische Ebene. So markierte es zum Beispiel 'Moonshot' und 'Red China' als Eigennamen. Dabei referenziert hier 'Moonshot' metaphorisch den Versuch der Amerikaner auf den Mond zu fliegen und ‘Red China‘ das kommunistische China[10]. Bemerkenswert ist auch, dass Laute als Eigennamen erkannt wurden, wie z.b. ‘di-dee-da da-dum‘ oder ‘ho‘. Die Genauigkeit des Modells im Vergleich zum Korrekturdatensatz betrug ~63,37%. Hier hatte spaCy wie NLTK Probleme damit alle Personennamen zu erkennen. Die Python Methode „def missing_person_percentage(correction_data, missing_Entitäten)“ ergab, dass spaCy ~23.29% aller Personennamen nicht erkannte. Probleme gab es auch bei der Erkennung von ‘MISC‘, wie ('MISC', 'Edsel') und ('MISC', 'Big Mac'). Denn MISC wurde von spaCy gar nicht erkannt, obwohl es die Fähigkeit hat es zu erkennen. Grund dafür, könnte aber auch die Limitation meines Vergleiches sein. Wenn ich code, dass spaCy nur vier NER-Tagsets erkennen soll, so werden die anderen NER-Tagsets ignoriert. SpaCy hat auch ein NER-Tagset ‘Product Names‘, welches zum Vergleich zwischen allen drei Packages nicht zum Einsatz

kommen konnte und Produktnamen wie ('MISC', 'Edsel') und ('MISC', 'Big Mac'). Enthalten sein könnten. Doch dafür müsste man spaCy an sich auf NER untersuchen.

5. Schlussfolgerung

In Anbetracht der Analyse der drei verschiedenen Modelle, Flair, nltk und spaCy zur Erkennung von Eigennamen aus Songtexten, lassen sich folgende Schlussfolgerungen ziehen. Zunächst einmal hatte das NLTK-Modell große Probleme mit der Erkennung von Entitäten, insbesondere von Personen und spezifischen Typen wie MISC oder LOC. Die hohe Überschätzungsquote von etwa 23,76% deutet darauf hin, dass viele falsche positive Entitäten erkannt wurden, was auf Probleme mit der Großschreibung von Wörtern in Songtexten zurückzuführen ist. Des Weiteren hatte das NLTK-Modell Schwierigkeiten Personen zu identifizieren, was zu einer niedrigen Genauigkeit von nur ~52,47% führte.
Das Flair-Modell zeigte eine verbesserte Leistung im Vergleich zum NLTK-Modell und insbesondere bei der Erkennung von MISC- und LOC-Entitäten. Obwohl die Überschätzungsquote immer noch hoch war mit ~13,86%, war die Genauigkeit mit ~63,61% besser als beim NLTK-Modell. Allerdings hatten auch hier einige Wörter aufgrund der Großschreibung Probleme verursacht.
Das spaCy-Modell erzielte die beste Überschätzungsquote mit nur ~1,98%, was auf eine genauere Erkennung von Entitäten hinweist. Allerdings hatte auch spaCy Schwierigkeiten bei der Erkennung bestimmter Entitäten, wie Personen und MISC. Die Genauigkeit lag bei ~63,37%, welches ähnlich wie beim Flair-Modell war.
Zudem gab auch Limitationen in diesem Vergleich, denn es wurden nicht alle NER-Tagsets betrachtet. Für den Zweck einer Vergleichbarkeit aller Modelle wurden zum Beispiel bei der Verwendung von spaCy nicht alle verfügbaren NER-Tagsets berücksichtigt. Dies könnte dazu führen, dass bestimmte Eigennamen und deren Entitätstypen nicht erkannt wurden, was die Vergleichbarkeit beeinträchtigt. Insgesamt zeigen die Ergebnisse, dass keines der Modelle perfekt in der Lage war, alle Entitäten korrekt zu identifizieren. Dies war auf die spezifischen Herausforderungen von Songtexten zurückzuführen, welche insbesondere metaphorische Ausdrücke und der Großschreibung von Wörtern im Text beinhaltet.

6. Literaturverzeichnis

Mein Code und die verwendeten Datensätze:

https://github.com/Flo-o/Digital-Trends/tree/5f14ff074e6bb4d993ac733536355b407707361a

Internetquellen:

Flair: GitHub - flairNLP/flair: A very simple framework for state-of-the-art Natural Language Processing (NLP)

SpaCy: spaCy · Industrial-strength Natural Language Processing in Python

Trained Models & Pipelines · spaCy Models Documentation

NLTK :: Natural Language Toolkit

Songtextquellen:

https://genius.com/Billy-joel-we-didnt-start-the-fire-lyrics

https://genius.com/Frank-sinatra-theme-from-new-york-new-york-lyrics

https://genius.com/The-clash-london-calling-lyrics

https://genius.com/Toto-africa-lyrics

https://genius.com/Larry-groce-junk-food-junkie-lyrics

https://genius.com/The-beatles-eleanor-rigby-lyrics

https://genius.com/Italobrothers-radio-hardcore-lyrics

https://genius.com/Janis-joplin-mercedes-benz-lyrics

https://genius.com/Gucci-mane-both-lyrics

https://genius.com/Billy-joel-piano-man-lyrics

https://genius.com/2416827

https://genius.com/Elvis-presley-viva-las-vegas-lyrics

Studien und Fachliteratur:

Christenson, Peter & Haan-Rietdijk, Silvia & Roberts, Donald & Bogt, T.F.M.. (2018). What has America been singing about? Trends in themes in the U.S. top-40 songs: 1960–2010. Psychology of Music. 47. 030573561774820. 10.1177/0305735617748205.

Craig, Clay & Flynn, Mark & Holody, Kyle. (2017). Name Dropping and Product Mentions: Branding in Popular Music Lyrics. Journal of Promotion Management. 23. 1-19. 10.1080/10496491.2016.1267679.

Harsh Patel. (2020). BioNerFlair: biomedical named entity recognition using flair embedding and sequence tagger. S.1.

Nltk: Bird, Steven & Klein, Ewan & Loper, Edward. (2009). Natural Language Processing with Python. p.281-286

Biemann, Chris (2022): Semantische Verarbeitung. In: Bieman, Chris / Heyer, Gerhard / Quasthoff, Uwe (2022): Wissensrohstoff Text. Eine Einführung in das Text Mining. Wiesbaden: Springer Vieweg. S. 106-110

7. Selbstständigkeitserklärung

Hiermit bestätige ich, Florian Rücker, die vorliegende Arbeit selbstständig und nur mit zugelassenen Hilfsmitteln angefertigt zu haben. Längere wörtliche Übernahmen aus anderen Werken habe ich unter Angabe der Quelle kenntlich gemacht.

Leipzig, 17.03.2024