Praktikumsdokumentation: Extraktion von nichtdeterministischen Streaming-String-Transducern aus einem bilingualen Korpus

Student: Alexander Jenke

Betreuer: Thomas Ruprecht, M. Sc.

Verantwortlicher Hochschullehrer: Prof. Dr.-Ing. habil. Heiko Vogler

***Zusammenfassung*—TODO**

Der entwickelte Code ist auf GitHub veröffentlicht.

I. Aufgabenstellung

Ziel des Praktikums ist die Extraktion eines gewichteten nichtdeterministischen Streaming-String-Transducer (NSST) aus einem bilingualen Korpus. Hierfür sind in der Aufgabenstellung folgende Aufgabenteile definiert:

1. Vorbereiten eines bilingualen Europarl-Korpus zur Weiterverarbeitung, wobei nicht zur Sprache gehörende Artefakte entfernt sowie Wörter und Satzzeichen in Token aufgeteilt werden sollen.
2. Trainieren eines Hidden-Markov-Model (HMM) auf die Quellsprache und anschließende Extraktion eines endlichen Automaten, welcher die Sprache erkennt.
3. Generieren eines totalen Alignments mittels einer geeigneten Methode
4. Extraktion und lesbares Speichern des NSST aus Korpus, endlichem Automaten und Alignments

II. Übersicht

Die einzelnen Teilschritte wurden in getrennten Skripten implementiert, da einzelne Verarbeitungsschritte rechenaufwendig und zeitintensiv sind. Zwischenergebnisse werden in separaten Dateien gespeichert, um diese in verschiedenen Experimenten wiederholt nutzen zu können.

Die Aufgabenteile 1,2 und 4 sind vollständig in Python3 implementiert, für Schritt 3 werden die Sprachdaten mit Python3 vorbereitet und die Alignments mit einer fertigen C++-Implementierung des fast_align-Algorithmus [1] generiert.

Wie in Abbildung 1 dargestellt, werden die von Europarl bezogenen Sprachdaten durch das *europarl_dataloader.py*-Skript entsprechend Aufgabenteil 1 vorbereitet, hierbei werden die Datensätze der Quell- und der Zielsprache unabhängig voneinander verarbeitet. Anschließend wird auf den aufbereiteten Europarldaten der Quellsprache mit dem *hmm_trainings-py*-Skript ein multinomiales Hidden Markov Model trainiert. Parallel werden durch das *alignment_createPairedFile.py*-Skript beide aufbereiteten Sprachdatensätzen in einer Datei kombiniert, aus welcher anschließend mittels dem *fast_align*-Algorithmus Alignments extrahiert werden. Abschließend werden das trainierte HMM, die dabei verwendet Zuordnung von Wörtern und Satzzeichen auf Token (Tokenization), die kombinierte Datei beider Sprachdatensätze sowie die generierten Alignments durch das *nsst_createRules.py*-Skript kombiniert, um die Regeln eines NSST zu erzeugen.

III. Europarl-Korpus

Grundlage für Extraktion der NSST-Regeln bildet ein bilingualer Sprachdatensatz des Europarl-Korpus der Version 7. Dieser in [2] vorgestellte Korpus enthält bilinguale Datensätze wurden aus den Protokollen der Tagungen des Europäischen Parlamentes extrahiert. Diese Protokolle sind in 11 Sprachen verfügbar und 20 der 110 möglichen Datensätze sind veröffentlicht. Die Protokolle im Zeitraum von April 1996 bis November 2011 wurden Blockweise aneinander ausgerichtet und in einzelne Sätze zerlegt. Daraus sind Satz-Parallele bilinguale Datensätze mit rund 60 Millionen Wörtern pro Sprache generiert worden [3].

Ein bilingualer Datensatz besteht aus zwei Textdateien, welche überwiegend einen Satz pro Zeile enthalten. Die Sätze beider Dateien sind anhand der Zeilennummer einander zugeordnet.

A. Sprachauswahl

Für die Bearbeitung des Praktikums wurde der deutsch-englische Korpus gewählt, um Zwischenergebnisse bewerten zu können, ohne Hilfe durch externe Übersetzungen zu benötigen. Hierbei ist deutsch als Quell- und englisch als Zielsprache ausgewählt worden. Da das in Teilaufgabe 2 trainierte HMM ausschließlich auf der Quellsprache trainiert wurde, beeinflusste ebenfalls die einfache Bewertbarkeit der Zwischenergebnisse diese Aufteilung in Quell- und Zielsprache. Unabhängig davon ist die Implementierung für eine Anwendbarkeit auf alle bilingualen Korpora konzipiert.

B. Aufbereitung

Entsprechend Teilaufgabe 1 werden die Europarl-Datensätze durch Ausführung des *europarl_dataloader.py*-Skripts zur

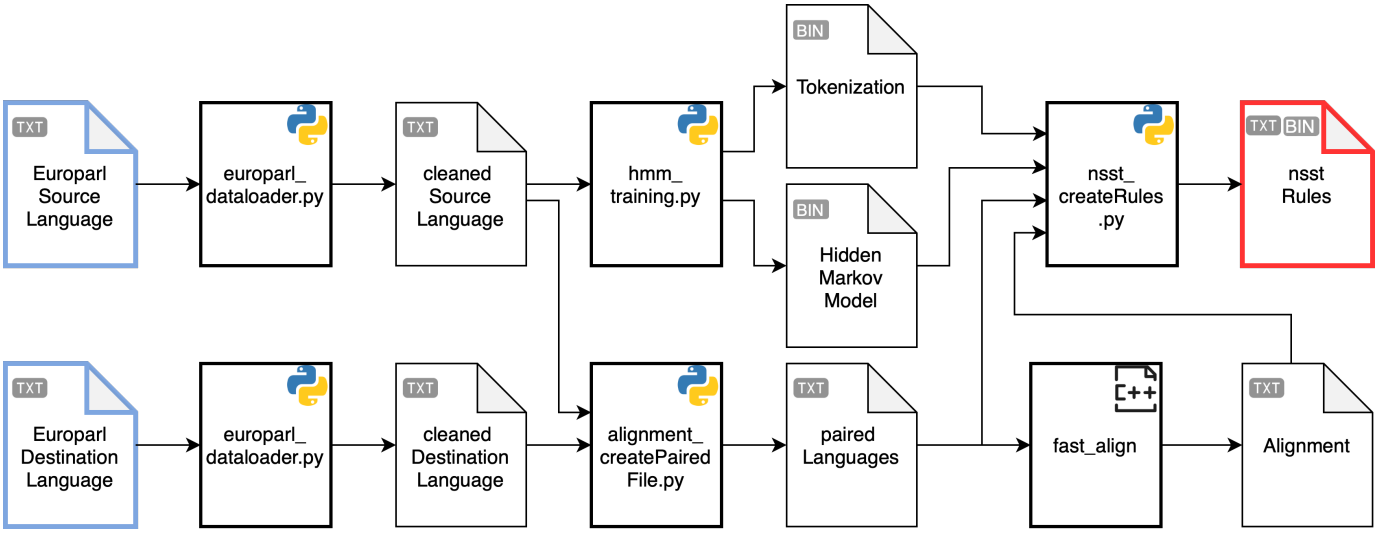

Abbildung 1. Datenfluss durch die einzelnen Verarbeitungsschritte.

Weiterverarbeitung vorbereitet. Hierfür lädt das Skript jeden Zeile der im Aufruf übergebenen Textdatei und bereinigt diese von allen Zeichen welche nicht explizit erwünscht sind.
Explizit erwünschte Zeichen sind:

- Buchstaben des Alphabets in Groß- und Kleinschreibung
- Umlaute in Groß- und Kleinschreibung, sowie ß
- französische Akzente (`, ´, ^) auf Vokalen in Groß- und Kleinschreibung
- Satzzeichen: Punkt, Komma, Ausrufe- und Fragezeichen
- Klammern: rund, geschweift und eckig
- Schrägstrich, umgedrehter Schrägstrich und Bindestriche
- Leerzeichen und Zeilenumbruch

Außerdem werden Verschiedene Bindestriche vereinheitlicht und Satzzeichen durch Leerzeichen von Wörtern oder anderen Satzzeichen abgegrenzt, um sicherzustellen, dass diese als einzelne Satzelemente erkannt werden. Abschließend werden alle aufeinander folgenden Leerzeichen auf ein Leerzeichen reduziert. Für die spätere Verwendung werden die aufbereiteten Zeilen in einer neuen Textdatei zwischengespeichert. Die Menge an Wörtern, Satzzeichen, Klammern und Schrägstrichen wird im Folgenden zusammenfassend Vokabeln genannt.

C. Tokenisierung

Wird das *europarl_dataloader.py*-Skript von einem anderen Skript eingebunden, stellt es neben der durch das Skript selbst genutzten Funktionalität zur Bereinigung der Sprache auch alle benötigten Funktionen zur Tokenisierung zur Verfügung.

Diese Funktionen werden durch das *hmm_training.py*-Skript benutzt, um die Vokabeln der aufbereiteten Quellsprache auf maschinenlesbare Tokens zu mappen. Diese maschinenlesbaren Tokens werden durch nicht-negative Integer-Zahlenwerte repräsentiert. Für die Erstellung eines Mappings werden unter Beachtung der Groß- und Kleinschreibung die jeweiligen Vorkommen der Vokabeln im verwendeten Datensatz gezählt. Anschließend werden die Vokabeln der Häufigkeit nach absteigend sortiert und mit 1 beginnend durchnummeriert. Abweichend davon kann ein Grenzwert festgelegt werden, ab welchem Vokabeln dem Sammel-Token mit der Nummer 0 zugewiesen werden, sobald deren Häufigkeit dem Grenzwert gleichen oder diesen unterschreiten. Die daraus resultierende Funktion ist surjektiv und wird im Folgenden Tokenisierung genannt. Wird kein Grenzwert definiert, ist die Tokenisierung bijektiv.

IV. Hidden-Markov-Model

Entsprechend Teilaufgabe 2 wird durch Ausführen des *hmm_training.py*-Skripts ein Hidden Markov Model unsupervised auf der Quellsprache trainiert. Das HMM kann als 5-Tupel definiert werden [4]:

$$
\begin{aligned}
HMM &= (Q_H, \Sigma_H, I_H, T_H, E_H) \quad \text{, mit} \\
Q_H &= \{q_0, q_1, ..., q_n\} \\
\Sigma_H &= \{0, 1, ..., m\} \\
I_H &= Q_H \to \mathbb{R}^+ \\
T_H &= Q_H \times Q_H \to \mathbb{R}^+ \\
E_H &= Q_H \times \Sigma_H \to \mathbb{R}^+
\end{aligned}
$$

Hierbei ist Q_H eine endliche Menge von Zustände, Σ_H eine endliche Menge von Token (Alphabet), I_H die Startwahrscheinlichkeit der Zustände, T_H die Übergangswahrscheinlichkeit von einem Zustand zum nächsten und E_H die Emissionswahrscheinlichkeit eines Token in einem Zustand [4].

Für das Training werden die Funktionen I_H, T_H und E_H des HMMs zufällig initialisiert und anschließend in mehreren Iterationen durch einen EM-Algorithmus auf die Sätze des Trainingsdatensatzes angepasst [5].

A. Architektur

Das HMM ist mittels der *MultinomialHMM*-Klasse des hmmlearn-Packages [6] implementiert. Dieser HMM-Typ emittiert in jedem Zustand entsprechend einer Gewichtung

einen Token. Es kann von jedem Zustand entsprechend einer Probabilistik in jeden Zustand gewechselt werden. Die Anzahl der Zustände und Token kann bei der Initialisierung beliebig gewählt werden.
Sei n_s die Anzahl der Zustände und n_t die Anzahl der Token, dann wird das HMM durch folgende drei Matrizen vollständig beschrieben:

startprob
: Die $1 \times n_s$-Matrix definiert die Startwahrscheinlichkeiten I der Zustände.

transprob
: Die $n_s \times n_s$-Matrix definiert die Übergangswahrscheinlichkeiten T zwischen Zuständen.

emissionprob
: Die $n_s \times n_t$-Matrix definiert die Emissionswahrscheinlichkeiten E der Token.

Zustände und Token sind durch nicht-negative Integer-Werte repräsentiert und lassen sich somit direkt als Index auf die Matrizen anwenden.

Um das Training zu beschleunigen, wurde eine Multi-Threading Variante des MultinomialHMM implementiert. Diese berechnet im E-Schritt des EM-Algorithmus in mehreren Threads parallel die logarithmische Wahrscheinlichkeit der einzelnen Sätze des Trainingsdatensatzes. Lediglich das Akkumulieren der für den M-Schritt benötigten Statistiken wird hierbei Sequenziell ausgeführt. Hierdurch wird die Rechenzeit bei Verdopplung der Threads nahezu halbiert, solange die Anzahl echter Kerne nicht überschritten wird.

Für jeden Eingabesatz kann mit dem HMM eine Wahrscheinlichkeit berechnet werden diese Eingabe mit einer bestimmten Zustandsreihenfolge zu lesen. Hierfür werden die Wahrscheinlichkeiten der Zustandsübergänge und die Emissionswahrscheinlichkeiten der Token im entsprechenden Zustand aufmultipliziert. Da diese Werte sehr klein werden sind sie üblicherweise auf einer logarithmischen Skala angegeben. Die Summe der Wahrscheinlichkeiten aller Möglichen Pfade über eine Eingabe wird im Folgenden Log-Likelihood genannt. Für die Log-Likelihood über mehrere Eingabesätze wird in der verwendeten Implementierung die Wahrscheinlichkeit der einzelnen Sätze aufaddiert. Die Zustandsreihenfolge für einen Eingabesatz mit der höchsten Wahrscheinlichkeit wird im Folgenden die beste Zusandsfolge genannt. Ziel des Trainings des HMMs ist es, die Log-Likelihood der Sätze des Trainingsdatensatzes zu maximieren.

B. Hyperparameter

Ausschlaggebend für den Erfolg beim Training des HMM ist die Wahl der Hyperparameter. Im Folgenden werden die zu wählenden Parameter sowie eventuelle Kriterien erläutert.

Die **train_step_size (TSS)** definiert in welcher Schrittgröße über die Sätze des Datensatzes gelaufen wird. Grundsätzlich sind die ersten 4096 Sätze der Evaluation des Modells vorbehalten, diese Sätze bilden den Testdatensatz. Alle verbleibenden Sätze können für das Training verwendet werden, jedoch kann die Größe dieses Datensatzes reduziert werden, indem nur jeder TSS-te Satz verwendet wird. Ausschlaggebend für die Wahl des TSS ist die Abwägung zwischen Speicher- und Rechenaufwand gegenüber einer ausreichenden Größe für einen repräsentativen Datensatz. Standardmäßig ist dieser Wert mit 20 implementiert. Der daraus resultierende Datensatz umfasst 95.806 Sätze mit 2.518.714 Vokabeln und 84.102 Token.

Der in Kapitel III erläuterte **Threshold** reduziert die Anzahl der Token, indem seltene Vokabeln im kumulativen Token 0 zusammenfasst werden. Dieser Parameter ist mit einem Standardwert von 4 implementiert, da bei stichprobenartiger Auswertung auf dem verwendeten deutschen Korpus bis zu diesem Wert überwiegend Eigennamen sowie viele Verben in konjugierter Form vorgekommen sind. Ab einem Wert von 5 wären vermehrt Verben in Grundform sowie beschreibende Vokabeln betroffen. Hierdurch reduziert sich die Zahl der Token auf 19.440. Somit werden 76,89% der möglichen Token im kumulativen Token 0 zusammengefasst, jedoch wirkt sich diese Reduktion nur auf 96044 Vokabelvorkommen aus, lediglich 3,81% der gesamten Trainingsdaten. Diese Werte beziehen sich auf einen TSS von 20, eine Übersicht über die Auswirkung verschiedener Threshold-Werte und weitere TSS ist im Anhang zu finden.

Während sich die Anzahlt der Token aus dem Datensatz ergeben, muss die **Anzahl der Zustände** des HMM festgelegt werden. Da dieser Wert die Größe des HMM bestimmt, sollte er mit Bedacht gewählt werden. Pro Zustand entwickeln sich während dem Training verschiedene Emissionswahrscheinlichkeiten der Token, wodurch semantisch und syntaktisch korrekte Sätze eine höhere Wahrscheinlichkeit erreichen. Da die Zustände somit indirekt die Kontext-Information des bisherigen Satzes abbilden, muss deren Anzahl ausreichen, um die Komplexität der Quellsprache abzubilden. Eine Obergrenze wird durch die zu trainierenden freien Parameter und die Zahl der Datenpunkte in den Trainingsdaten gebildet. Sei n_s die Anzahl der Zustände und n_t die Anzahl der Token, so lässt sich die Anzahl freier Parameter n_param wie folgt berechnen:

$$n_{param} = (n_s - 1) + [n_s * (n_s - 1)] + [n_s * (n_t - 1)]$$

Hierbei ergibt sich der erste Summand aus den Startwahrscheinlichkeiten I. Da diese aufsummiert eins ergeben, lässt sich der letzte Wert aus allen vorherigen bestimmen, und es bleiben $(n_s - 1)$ freie Parameter.
Der zweite Summand ergibt sich aus den Transitionswahrscheinlichkeiten T. Pro Zustand summieren sich die Übergangswahrscheinlichkeiten zum nächsten Zustand ebenfalls zu eins auf. Somit bleiben für jeden Zustand $(n_s - 1)$ freie Parameter, also gesamt $[n_s * (n_s - 1)]$.
Der dritte Summand ergibt sich aus den Emissionswahrscheinlichkeiten E. Pro Zustand Summieren sich die Wahrscheinlichkeiten aller Token zu eins auf und es bleiben $[n_s * (n_t - 1)]$ freie Parameter.
Ist die Anzahl der freien Parameter größer als die Zahl der

Datenpunkte, können nicht alle Parameter bestimmt werden und die Lösung ist degeneriert [7]. Somit führt eine größere Anzahl an Zuständen ab diesem Punkt lediglich zu einen höheren Rechenaufwand ohne neue Informationen in den Zuständen abzubilden.
Die Anzahl der Datenpunkte im Datensatz entspricht der Anzahl der Vokabeln. Ein TSS von 20 führt zu 2.518.581 Vokabeln, ein Threshold von 4 zu 19440 Token. Damit liegt für diese Parameter die Obergrenze der Zustände bei 128.
In der Implementierung wurde das HMM mit 10, 54, 100 und 200 Zuständen trainiert. Hierbei entsprechen die 54 Zustände den in [8] definierten 54 POS-Tags. Mit steigender Anzahl an Zuständen, hat sich beim Training eine steigende Wahrscheinlichkeit der Test- und Trainingsdaten gezeigt, jedoch ist auch die Berechnungszeit gestiegen. Somit muss bei der Wahl dieses Parameters, unter Beachtung der erläuterten Grenzen, zwischen guter Repräsentation des Datensatzes und Rechenaufwand abgewogen werden.

Abschließend muss für das Training noch bestimmt werden, in wie vielen **Iterationen** das HMM auf den Datensatz angepasst werden soll. In jeder Iteration wird der komplette Datensatz im E-Schritt des EM-Algorithmus verarbeitet und anschließend das HMM im M-Schritt angepasst. Da die Anpassungen in kleinen Schritten erfolgt wird eine ausreichende Anzahl an Iterationen benötigt um gute Ergebnisse zu erzielen. Jedoch kann es bei zu vielen Schritten zu einer übermäßigen Anpassung (Overfitting) auf die Trainingsdaten kommen, sodass das Model an Allgemeingültigkeit verliert. Dies wird in einer fallenden Wahrscheinlichkeit der Testdaten bei steigender Wahrscheinlichkeit der Trainingsdaten deutlich. In der Implementierung haben sich 100 Iterationen bewährt, da zu diesem Zeitpunkt die Wahrscheinlichkeiten nicht mehr stark stiegen und es noch nicht zu Overfitting kam.

C. endlicher Automat

Wie in [4] beschreiben wird, kann aus einem HMM ein gewichteter endlicher Automat abgeleitet werden, welcher die selbe Quellsprache erkennt. Hierfür wird der Automat (WFSA) wie folgt definiert:

$$
\begin{aligned}
HMM &= (Q_H, \Sigma_H, I_H, T_H, E_H) \\
WFSA &= (Q_A, \Sigma_A, \delta_A, I_A, P_A, F_A) \\
Q_A &= Q_H \cup \{q_{-1}\} \\
\Sigma_A &= \Sigma_H \\
\delta_A &= \{(q_{-1}, a, q) \colon I(q) \neq 0 \wedge E(q,a) \neq 0\} \\
\delta_A &= \{(q', a, q) \colon T(q', q) \neq 0 \wedge E(q,a) \neq 0\} \\
I_A(q_{-1}) &= 1 \\
I_A(q) &= 0 \\
P_A(q_{-1}, a, q) &= I(q) \cdot E(q,a) \\
P_A(q', a, q) &= T(q', q) \cdot E(q,a) \\
F_A(q_{-1}) &= 0 \\
F_A(q) &= 1 \\
&\text{wobei } \forall q, q' \in Q_H, a \in \Sigma_H
\end{aligned}
$$

Hierbei ist Q_A die Menge der Zustände des HMM, erweitert um den Startzustand q_{-1}. Das Alphabet Σ_A gleicht dem HMM. Die Übergänge δ_A lesen ein Zeichen aus dem Alphabet und verbinden Zustände deren Transitionswahrscheinlichkeit T_H im HMM nicht 0 ist, sowie den Startzustand mit allen Zuständen deren Startwahrscheinlichkeit I_H nicht 0 ist. Die Anfangsgewichte I_A definieren den Startzustand als einzig möglichen Anfangszustand, die Endgewichte F_A definieren alle Zustände außer den Startzustand als Finalzustände. Die Übergangsgewichte P_A sind das Produkt aus der Transitionswahrscheinlichkeit T_H mit der Emissionswahrscheinlichkeit E_H des gelesenen Tokens im Zielzustand, wobei die Transitionswahrscheinlichkeit vom Startzustand zu einem anderen Zustand der Startwahrscheinlichkeit I_H entspricht. Zu beachten ist, dass dieser Automat die Wahrscheinlichkeit der gelesenen Eingabe berechnet. Daher repräsentieren die Übergangsgewichte eine Probabilität der Nutzung dieser Kante, für eine Interpretation als Kosten müssen die Werte von eins subtrahiert werden.

Der resultierende Automat stellt bei der NSST-Regel-Extraktion die Zustandsreihenfolge für die Verarbeitung des Quell-Satzes bereit. Alternativ dazu kann direkt mit dem HMM eine ideale Zustandsfolge für den zu übersetzenden Satz ermittelt werden und diese analog zur WFSA-Extraktion um einen Startzustand q_{-1} ergänzt werden.

V. Alignment

Die Alignments verbinden die Satzpositionen der Quell- und Zielsätze entsprechend ihrer semantischen Korrelation und geben somit Auskunft, welche Vokabel im Quellsatz, welche Vokabeln des Zielsatzes erzeugt. Für eine Übersetzung muss jede Satzposition des Zielsatzes von genau einer Position des Quellsatzes erzeugt werden, diese Eigenschaft heißt total.
Sei X die Menge aller Satzpositionen des Quellsatzes und Y die Menge aller Satzpositionen des Zielsatzes, dann ist die Funktion $f(x) = y, \exists! x \in X, \forall y \in Y$ ein totales Alignment.

Diese totalen Alignments werden mittels des *fast_align*-Algorithmus [1] generiert. Dieser passt durch den EM-Algorithmus ein lexikalisches Übersetzungsmodell an die eingegebenen Satzpaare an [1] und gibt am Ende ein Alignment pro Satz aus. Ein Alignment besteht aus einer Zeile und enthält mehrere Satzpositionspaare welche durch ein Leerzeichen getrennt sind. Diese Satzpositionspaare enthalten jeweils eine Satzposition der Quell- und Zielsprache, getrennt durch einen Bindestrich.

Die benötigte Eingabedatei wird durch Ausführen des *alignment_createPairedFile.py*-Skripts generiert und enthält in jeder Zeile den Quell- und Zielsatz, getrennt durch das von *fast_align* geforderte Trennzeichen „ ||| “.

Wird der *fast_align*-Algorithmus mit den Optionen -d, -o, -v und -N ausgeführt generiert dieser ein totales Alignment. Hierbei sind die ersten drei Optionen durch die Entwickler ausdrücklich empfohlen und führen zur Verwendung einer Dirichlet-Verteilung und der Bevorzugung von Verknüpfungen nahe der monotonen Diagonale. Die N-Option verbietet die

Nutzung des null-Wortes und führt zu einem totalen Alignment. Bei nicht-Benutzung der vierten Option müssen nicht alle Satzpositionen der Zielsprache einem Satzpositionspaar zugeordnet werden.

VI. NICHTDETERMINISTISCHER STREAMING-STRING-TRANSDUCER

Abschließend wird entsprechend Teilaufgabe 4, durch Ausführen des *nsst_createRules.py*-Skripts, der NSST extrahiert. Hierfür werden das trainierte HMM, die verwendet Tokenization, die kombinierte Datei beider Sprachdatensätze sowie das dazugehörige Alignment verarbeitet und für jeden Satz die zur Übersetzung benötige Menge an Transitionen generiert. Der damit definierte NSST kann Sätze der Quellin die Zielsprache übersetzen.

A. Span

Anhand der Alignments wird ermittelt welche Vokabel der Quellsprache welche Vokabeln der Zielsprache generiert. Für jede Position im Quellsatz lässt sich bestimmen welche Satzpositionen im Zielsatz bereits generiert wurden. Diese Funktion welche eine Quellsatzposition auf eine Menge zusammenhängender Zielsatzpositionen abbildet, nennen wir im Folgenden $Span$. Sei G eine Funktion, die für eine Quellsatzposition die Menge der durch diese Wortposition generierten Zielsatzpositionen gibt, $\{p_0, p_1, ..., p_n\}$ die Menge der Quellsatzpositionen und $\{q_0, q_1, ..., q_m\}$ die Menge der Zielsatzpositionen, dann lässt sich $Span$ wie folgt definieren:

$$
\begin{aligned}
Span(p_i) &= \{Span(p_{i-1}) \cup G(p)\} \quad \forall p \in \{p_1, p_2, .., p_n\} \\
Span(p_0) &= \{G(p_0)\} \\
Span(p_n) &= \{q_0, q_1, ..., q_m\}
\end{aligned}
$$

Die Menge der Zielsatzpositionen einer Quellsatzposition lassen sich in Submengen teilen, sodass jede Submenge alle Positionen zwischen der keinsten und größten enthaltenen Satzposition enthält. Diese Submengen lassen sich durch eben diese kleinste und größte enthaltene Satzposition beschreiben.

B. NSST

Die Autoren in [9] beschreiben den nichtdeterministischer Streaming-String-Transducer als einen ein Automaten, welcher eine Eingabe ließt und für diese auf der Ausgabe eine Übersetzung generiert. Hierfür besitzt er ein Eingabe-Band und eine endliche Menge von String-Variablen über dem Ausgabealphabet aus welcher am Ende die Übersetzung abgelesen werden kann. Diese Menge wird im Folgenden Register genannt. Der Automat läuft einmal von links nach rechts über die Eingabe. In jedem Schritt ließt er ein Symbol auf der Eingabe, ändert seinen Zustand und aktualisiert mittels einem *Copyless-Assignment* gleichzeitig alle String-Variablen im Register. Die rechte Seite dieser Assignments besteht aus einer Aneinanderreihung von String-Variablen und Zeichen des Ausgabealphabets, wobei jede String-Variable maximal einmal verwendet werden darf.

Der implementierte NSST lässt sich wie Folgt definieren:

$$
\begin{aligned}
NSST &= (Q_N, \Sigma_N, \Gamma_N, X_N, E_N, q_{-1}, F_N) \\
Q_N &= Q_A \\
\Sigma_N &= \Sigma_A = \Sigma_H = \{0, 1, ..., m\} \\
\Gamma_N &= \{0, 1, ..., o\} \\
X_N &= \{x_1, x_2, ...x_p\} \\
x_i &\in \Gamma_N^* \qquad \forall x_i \in X_N \\
A &\colon X_N \to (X_N \cup \Gamma_N) \\
E_N &\subseteq (Q_N \times \Sigma_N \times A \times Q_N) \\
count &\colon E_N \to \mathbb{N} \\
F_N &\colon Q_N \to (X_N \cup \Gamma_N) \\
F_N(q) &= x_1 \qquad \forall q \in Q_N
\end{aligned}
$$

Q_N ist die endliche Menge an Zuständen und gleicht den Zuständen des aus dem HMM extrahierten WFSA. Σ_N und Γ_N sind die Ein- und Ausgabealphabete. Sie werden durch die endlichen Mengen der Token von Ziel- und Quellsprache repräsentiert. Das Eingabealphabet gleicht dem Alphabet des HMM und des WFSA. Das Ausgabealphabet wird durch den NSST das erste Mal verwendet, hierfür wird auf den Sätzen der Zielsprache ein Alphabet ohne Threshold generiert. X_N ist eine endliche Menge von String-Variablen über dem Ausgabealphabet. Die Länge wird in der Implementierung nicht explizit definiert, sondern ergibt sich aus den extrahierten Regeln. A ist eine Menge von Copyless-Assignments welche Registerinhalte durch Aneinanderreihungen von Registerinhalten und Zielsprachtoken definiert. E_N ist die Menge möglicher Transitionen. Eine Transition besteht aus dem aktuellen Zustand, dem gelesenen Token, dem angewendete Assignment und dem daraus resultierenden neuen Zustand. Die Funktion $count$ ordnet jedem Assignment eine natürliche Zahl zu. Dieser Wert gibt an, wie oft das Assignment aus den Daten extrahiert wurde und wird zur Berechnung der Wahrscheinlichkeit des Assignments benötigt. q_{-1} ist der Startzustand und gleicht dem Startzustand des WFSA. F_N ist eine partielle Ausgabefunktion, welche nach dem Lesen der Eingabe abhängig vom Zustand des NSST aus dem Register die Ausgabe generiert. In der Implementierung wird unabhängig vom Zustand immer die erste String-Variable des Registers als Ausgabe verwendet.

C. Extraktion

Die Hauptaufgabe bei der Extraktion des NSST besteht darin, alle notwendigen Transitionen zu ermitteln. Hierfür wird aus der Zustandsreihenfolge, mit welcher der WFSA den Quellsatz mit der höchsten Wahrscheinlichkeit liest, und dem Alignment zwischen Quell- und Zielsatz für jeden Zustandsübergang bestimmt, wie das Register aktualisiert werden muss. Daraus ergeben sich die benötigten Transitionen.

Sowohl der NSST als auch der WFSA, beziehungsweise das HMM, verwenden die Token der Quell- bzw. Zielsprache als Alphabete. Daher müssen im ersten Schritt die Sätze der Quell- und Zielsprachen mittels der übergebenen Tokenisierungen in eine automatenlesbare Form gebracht werden.

Anschließend wird die Zustandsreihenfolge für den Quellsatz ermittelt. Da das implementierte HMM bereits eine Funktion zur Verfügung stellt, die wahrscheinlichste Zustandsreihenfolge für eine Tokenfolge zu ermitteln, wurde in der Implementierung darauf verzichtet diese Reihenfolge mittels dem WFSA zu ermitteln. Stattdessen wird, analog zur Extraktion des WFSA aus dem HMM, die durch das HMM errechnete Zustandsfolge um den einheitlich Startzustand q_{-1} ergänzt. Unter der Annahme, dass der extrahierte WFSA und das HMM die selbe Sprache erkennen, führt die Verwendung beider Automaten auch zur selben Zustandsreihenfolge.

Eine Transition enthält neben dem Zustandsübergang und dem entsprechenden Token der Quellsprache auch ein Assignment mittels welchem das Register aktualisiert wird. In der Implementierung wird dieses generiert, indem alle bereits generierten, direkt aufeinander folgenden Satzpositionen in einer String-Variable konkateniert werden. Nicht konkatenierbare Satzteile werden im Register nach aufsteigender Satzposition sortiert. Diese Mengen von aufeinander folgenden Satzpositionen entsprechen dem Span der gelesenen Satzposition.

Für jedes Satzpaar wird zu Beginn ein leeres Register initialisiert. Anschließend wird für jeden Zustandsübergang im Register der Span der gelesenen Satzposition konstruiert. Hierfür werden bereits im Register vorhandene Satzpositionsmengen kopiert und diese mit den fehlenden, neu zu generierenden Token der Zielsprache konkateniert. Das Assignment gibt an, in welcher String-Variable welche String-Variablen und welche generierten Token in welcher Reihenfolge konkateniert werden. Da jede Satzposition des Zielsprache pro Span nur einmal vorkommen kann, sind die auf diese Weise generierten Assignments implizit copyless.

Ein Beispiel für eine Assignment-Extraktion ist im Anhang zu finden.

VII. Diskussion

A. Europarl-Korpus

Während der Arbeit mit dem Europarl-Korpus ist aufgefallen, dass die maschinengenerierte Zuordnung der Sätze der Quell- und Zielsprache zueinander teilweise fehlerhaft ist. Problematisch ist hierbei, wenn ein Satz in Sprache A einen Nebensatz enthält und diese in Sprache B durch zwei Sätze übersetzt wurde. Da im Europarl-Korpus nur eine 1-zu-1 Zuordnung zwischen Sätzen möglich ist, wird der Satz aus Sprache A nur teilweise Übersetzt und der Verbleibende Satz in Sprache B wird einer leeren Zeile oder einem einzelnen Punkt in Sprache A zugeordnet.

Hiervon sind im verwendeten Trainingsdatensatz (TSS=20) 303 der gesamt 95806 Sätze betroffen, das entspricht lediglich 0.32% der Sätze. Daher wurde dieses Problem nicht weiter behandelt. Der Einfluss dieser fehlerhaften Satzzuordnungen auf die Alignments und den letztendlich extrahierten NSST sollte in folgenden Arbeiten weiter untersucht werden.

B. kumulativer Token 0

Um die Anzahl der Token stark zu reduzieren wird bei der Erstellung der Tokenisierung der Quellsprache ein Threshold von 4 verwendet. Dieser Wert wurde durch eine stichprobenartige Analyse verschiedener Threshold-Werte ermittelt, somit ist nicht ausgeschlossen, dass auch semantisch relevante Worte fälschlicherweise im kumulativen Token zusammengefasst werden.

Das Gebiet anderer Verfahren Token zusammenzufassen bietet viel Potential für weitere Forschungsarbeiten. So ist es beispielsweise denkbar semantisch weniger interessante und selten vorkommende Worte gezielt anhand der Wortart oder alle konjugierten Formen eines Verbs zu einem Token zusammenzufassen.

C. Alignments

Die in dieser Arbeit durch den fast_align-Algorithmus generierten Alignments sind durch die verwendeten Parameter zwingend total. Jedoch führen die in Kapitel VII-A erläuterten Probleme des Europarl-Korpus unausweichlich zu fehlerhaften Alignments.

Die Qualität der extrahierten Alignments konnte im Rahmen dieser Arbeit nicht beurteilt werden. Diese sollte jedoch wegen ihrem direkten Einfluss auf die extrahierten Assignments untersucht und gegebenenfalls optimiert werden.

D. extrahierte Assignments

Für die Trainingsdaten(TSS=20) werden für 54, 100 als auch 200 Zustände jeweils eine Millionen verschiedene Assignments generiert. Hiervon lesen jeweils knapp 5% den kumulativen Token 0. Für jeden anderen Token gibt es im Schnitt 17 Regeln die unterschiedliche Token der Zielsprache generieren. Im Schnitt wurde jede Regel 2 mal generiert. Die genauen Zahlen sind im Anhang zu finden.

Um die extrahierten Transitionen zu evaluieren wurden die Validierungsdaten mittels der NSST übersetzt. Die Funktionalität hierfür ist im nsst_translate.py-Skript implementiert. Um den Rechen- und Speicheraufwand der Übersetzung zu reduzieren wurde lediglich die beste Transitionsfolge betrachtet. Hierfür wird bei jedem gelesenen Token für jeden erreichbaren Zustand lediglich das Register mit der besten Wahrscheinlichkeit gespeichert. Somit müssen bei der Verarbeitung eines gelesenen Token die möglichen Transitionen auf maximal die Menge der Zustände angewendet werden. Es wurde auch betrachtet inwieweit die Auswahl anwendbarer Transitionen eine Rolle spielt, so wurden einmal alle Transitionen mit passendem Zustand und Token gewählt und einmal lediglich die Transitionen die neben korrektem Zustand und Token auch exakt die Menge der String-Variablen im übergebenen Register verarbeiten.

Als Übersetzung wurde der Endzustand mit der Höchsten Wahrscheinlichkeit gewählt. Alle anderen Endzustände wurden ignoriert. Alle Sätze für welche eine Übersetzung gefunden werden konnte, wurden mittels dem BLEU-Score bewertet. Hierbei zeigt sich, wie in Tabelle I zu sehen, dass eine Eingrenzung der anwendbaren Transitionen das Ergebnis verbessern. Auch steigt der Score mit steigender Anzahl von Zuständen. Alle Ergebnisse sind auf der Scala von 0-1 bei einem maximal erreichen Ergebnis von 0,1 als schlecht zu

bewerten. Inwieweit dieses Ergebnisses durch Verbesserungen in den bereits erläuterten Bereichen optimiert werden kann, muss noch in weiterer Forschung erarbeitet werden.

	nur Regeln mit passender Anzahl String-Variablen	
#Zustände	ja	nein
54	0,0725	0,0616
100	0,0816	0,0663
200	**0,1081**	0,0778

Tabelle I
BLEU-SCORE AUF VALIDIERUNGSDATEN

LITERATUR

[1] C. Dyer, V. Chahuneau, and N. A. Smith, "A simple, fast, and effective reparameterization of ibm model 2," in *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2013, pp. 644–648.

[2] P. Koehn, "Europarl: A parallel corpus for statistical machine translation," in *MT summit*, vol. 5. Citeseer, 2005, pp. 79–86.

[3] "European parliament proceedings parallel corpus 1996-2011," https://www.statmt.org/europarl/, abgerufen: 2020-04-23.

[4] A. Becker and G. Jähnig, "Hidden-markov-modelle als gewichtete endliche automaten," 2008.

[5] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[6] "hmmlearn-dokumentation: Multinomialhmm," https://hmmlearn.readthedocs.io/en/latest/api.html#multinomialhmm, abgerufen: 2020-04-24.

[7] A. Allahverdyan and A. Galstyan, "Comparative analysis of viterbi training and maximum likelihood estimation for hmms," in *Advances in Neural Information Processing Systems*, 2011, pp. 1674–1682.

[8] M. Durrell, P. Bennett, S. Scheible, and R. J. Whitt, "The germanc corpus," *Manchester: School of Languages, Linguistics and Cultures*, 2012.

[9] R. Alur and J. V. Deshmukh, "Nondeterministic streaming string transducers," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2011, pp. 1–20.

VIII. ANHANG

TSS = 1	**Vokabeln**		**Vokabel-Vorkommen**	
Threshold	#	%	#	%
All	404387	-	50220274	-
1	207333	51,27	207333	0,41
2	260282	64,36	313231	0,62
3	286078	70,74	390619	0,78
4	301604	74,58	452723	0,90
5	312604	77,30	507723	1,01
6	320840	79,34	557139	1,11
7	327183	80,91	601540	1,20
8	332410	82,20	643356	1,28
9	336794	83,29	682812	1,36
10	340479	84,20	719662	1,43

Tabelle II
AUSWIRKUNG VERSCHIEDENER THRESHOLD-WERTE AUF DEN DEUTSCHEN KORPUS MIT TSS=1

TSS = 20	**Vokabeln**		**Vokabel-Vorkommen**	
Threshold	#	%	#	%
All	84102	-	2518714	-
1	45184	53.73	45184	1.79
2	56052	66,65	66920	2,66
3	61368	72,97	82868	3,29
4	**64662**	**76,89**	**96044**	**3,81**
5	66931	79,58	107389	4,26
6	68624	81,60	117547	4,67
7	70001	83,23	127186	5,05
8	71053	84,48	135602	5,38
9	71997	85,61	144098	5,72
10	72752	86,50	151648	6,02

Tabelle III
AUSWIRKUNG VERSCHIEDENER THRESHOLD-WERTE AUF DEN DEUTSCHEN KORPUS MIT TSS=20

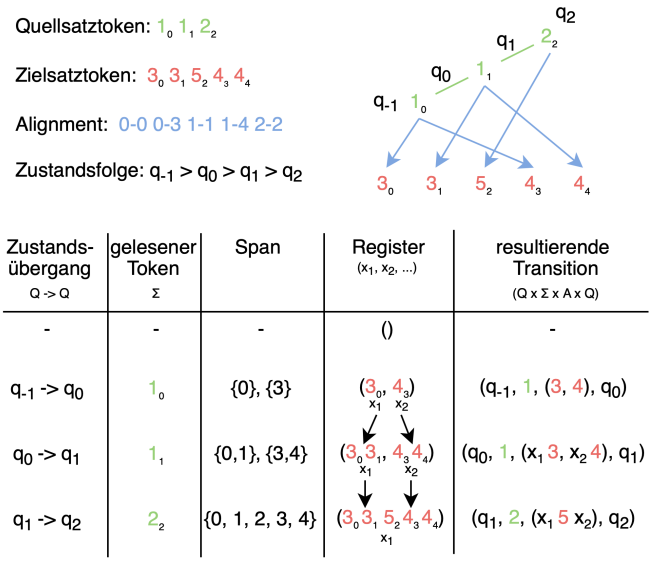

Abbildung 2. Beispiel einer Assignment-Extraction

# Zustände	**# Transitionen**	**∅ versch. generierte Token pro gelesenem Token (mean±std)**	**∅ count**
Alle Transitionen			
54	1.142.276	18,999±352,684	2,205
100	1.206.374	18,998±352,516	2,088
200	1.271.216	19,021±352,996	1,981
Transitionen ohne kommutativen Token 0			
54	1.083.927	17,141±239,343	2,235
100	1.146.700	17,140±239,119	2,113
200	1.209.428	17,163±239,817	2,003

Tabelle IV
STATISTIK DER EXTRAHIERTEN NSST-TRANSITIONEN