Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

Independent Coursework

Angewandte Informatik HTW Berlin
Wintersemester 20/21

Entitäten Erkennung in historischen Bundestagsprotokollen

Prof. Dr. Thomas Hoppe

Truong An Nguyen
s0575065

1 Einleitung

Heutzutage werden die Plenarsitzungen und Diskussionen im Bundestag mitgeschrieben. Wenn die Abgeordneten diskutieren, wird jedes gesprochene Wort von Stenografen dokumentiert. Dabei versuchen die Stenografen alle Wortmeldungen, sonstige Äußerungen und Interaktionen im Plenarsaal zu erfassen, was um sie herum passiert: Wer was sagt, wer klatscht, wer dazwischenruft, wer lacht, wer einer Aussage widerspricht.

Diese Mitschriften sind zu umfangreich und schwer beim Lesen oder es gibt viele Schwierigkeiten zur Informationsextraktionen. Zum Vereinfachen liegen Sitzungsprotokolle als XML-Dateien vor, daher sind die spezifischen Informationen daraus leichter zu ziehen. Jedoch wurden die Plenarprotokolle der 18. Wahlperioden noch nicht sauber getaggt und unklar in XML-Wrapper ausgezeichnet, deshalb ist es schwierig, ihren Inhalt zu analysieren. Um dieses Problem zu lösen, verwende ich zunächst das Conditional Random Fields Modell (CRFs) zur Segmentierung und Klassifikation von Wörtern. Dann benutze ich die Technik sogenannte Chunking zur Bearbeitung von CRFs-Ergebnissen und zur Erstellung von XML-Datei, die eine saubere und ordentliche Struktur hat.

2 Grundbegriffe

A. Conditional Random Fields

Das Conditional Random Fields (CRFs) ist ein ungerichteter, probabilistischer Modelltyp, der im maschinellen Lernen und einem Teilgebiet der künstlichen Intelligenz eingesetzt wird. CRFs hat nicht nur alle Vorteilen von Maximum-entropy Markov Model(MEMM) **[1]** inhärent, sondern auch die Fähigkeit, das Problem „Label Bias“ zu lösen. Das Problem „Label Bias“ bedeutet: Wenn ein Zustand sich zum einen nächsten Zustand verändert, achten diese zwei Zustände nur auf einander und es ist irrelevant, wie die anderen Zustände davor und danach aussehen**[2]**. Theoretisch kann das CRF-Modell als ein Modell unter Verwendung von unnormalisierten Übertragungswahrscheinlichkeitsverteilung betrachtet werden. Diese unnormalisierte Natur ermöglicht den Übergangsschritten, verschiedene wichtige Werte zu erhalten. Daher kann jeder Zustand die Wahrscheinlichkeit von nächsten Zuständen erhöhen oder verringern und es ist noch sichergestellt, dass die endgültige Wahrscheinlichkeit der gesamten Folge gültig und gut erfüllt ist.

CRFs erhält eine Sequenz als Eingabe und gibt, nach Bearbeitung, auch eine gleich lange Sequenz zurück. Im Unterschied zu den anderen Modellen kann CRFs an jeder Stelle auf die komplette Information der Eingabesequenz zugreifen, wohingegen ein MEMM nur die aktuelle Eingabe sieht. Hierdurch können komplexe Merkmalsmengen verwendet werden. Daher wird

CRFs anders als MEMM und Hidden Markov Model (HMM), sehr erfolgreich in vielen echten Probleme zur Segmentierung von Sequenzen verwendet **[2]**.

C.1 Definition

Das Ziel des CRFs ist die Klassifikation einer Beobachtungssequenz X. Dabei wird die Beobachtung $x_i \in X$ einer Labelsequenz $y_i \in Y$ zugewiesen. Und die bedingte Wahrscheinlichkeit für die Labelsequenz wird laut der Beobachtungssequenz X berechnet und ist abhängig von der Beobachtungssequenz X auch.

Wenn wir den ungerichteten Graphen G = (V,E) global auf X konditionieren, wenn die Werte von Zufallsvariablen in X fest oder gegeben sind, folgen alle Zufallsvariablen in Menge Y der Markov-Eigenschaft $p(Y_v \mid X, Y_u, u \neq v, \{u,v\} \in V) = p(Y_v \mid X, Y_u, (u,v) \in E)$. Diese Formel bedeutet, dass die Wahrscheinlichkeit von jeder Zufallsvariable Y_v bei X und von allen anderen $Y_{\{u \mid u \neq v,\ \{u,v\} \in V\}}$ gleich genauso wie die Wahrscheinlichkeit von Zufallsvariable Y_v bei X und den anderen Zufallsvariablen, die mit den benachbarten Knoten von dem Knoten Y_v in G entsprechen, ist. Dann ist (X, Y) ein ,,Conditional random field".

Nimmt man an, dass X = (X_1, X_2, ... X_n) und Y = (Y_1, Y_2, ... Y_n) gegeben sind, hat der Graph G die folgende Form(Abbildung 1).

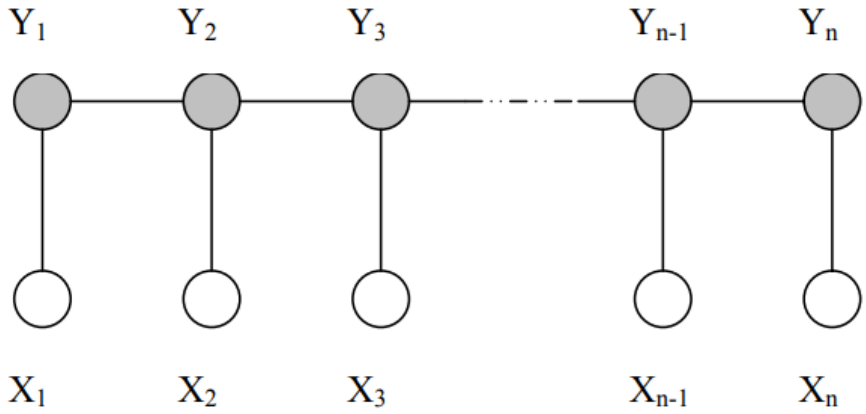

Abbildung 1: Graph eines CRFs

Angenommen, C sei der vollständige Satz von Untergraphen von G, so ist der vollständige Untergraph tatsächlich nur ein Knoten oder eine Kante des Graphen G. Da liegt daren, dass es sich bei dem Graphen G um eine first-order Markov-Kette handelt. Wir wenden die Ergebnisse von Hammerley Clifford **[3]** auf Markov Random Field an, dann wird die bedingte Wahrscheinlichkeit einer Labelsequenz Y und einer Beobachtungssequenz X berechnet.

$$p(y \mid x) = \prod_{(y_{i-1},\, y_i) \in C} \Psi_i\,(y_{i-1},\, y_i \mid x) \quad (1.1)$$

Wobei Ψ_i die Potentialfunktion ist und $(y_{i-1},\, y_i)$ die Kante mit Label y_i in C ist.

John Lafferty zeigte die Potentialfunktion von dem CRF Modell in der folgenden Form **[2]**.

$$\Psi_i\,(y_{i-1},\, y_i \mid x) = \exp(\Lambda_i(\,y_{i-1},\, y_i \mid x\,))$$

$$\Lambda_i(\, y_{i-1},\, y_i \mid x\,) = \sum_k \lambda_k \, \mathrm{f}_k(y_{i-1},\, y_i, x) + \sum_k \mu_k \, \mathrm{g}_k(\, y_i, x) \qquad (1.2)$$

Hierbei $\mathrm{f}_k(y_{i-1},\, y_i, x)$ ist das Attribut von den Labels von den Positionen i und i-1 in der Labelsequenz Y. $\mathrm{g}_k(\, y_i,\, x)$ ist das Labelattribut an der Position i in der Labelsequenz Y. Des weiteren wird der Wert beider Attribute von den Trainingsdaten entnommen und mit einer Konstante versehen(Abbuldung 2). Hier sind λk und μk konstante Lagrange-Multiplikatoren von den Attributen fk und gk. Als λk, μk werden die Wichtigkeiten von den Attributen fk und gk in der Datenkette bestimmt.

$f_i =$ { 1 falls x_{i-1}= „(“ und x_i= „Beifall“ und y_{i-1}= „BBracket“ und y_i= „MOOD“ ; 0 umgekehrt }

$g_i =$ { 1 falls x_i= „Beifall“ und y_i= „MOOD“ ; 0 umgekehrt }

Abbildung 2: Beispiel von f_i und g_i.

CRFs müssen nicht alle möglichen Beobachtungssequenzen X aufzählen, und daher können diese Potentialfunktionen direkt nach Bedarf aus einer gegebenen Trainings- oder Testbeobachtungssequenz x und dem Parametervektor θ berechnet werden. Dann ist die Normalisierung (Partitionsfunktion) $Z_\theta(x)$ der Eintrag von (start = y_0, y_{n+1} = stop) dieser Potentialfunktionen **[2]**.

$$\mathrm{Z}_\theta\,(x) = \prod_{start}^{stop} \Psi_i\,(x) \qquad (1.3)$$

Wir verwenden die Formel, doe on der Formulierung in 1.3 zu sehen ist, sodass sich die bedingte Wahrscheinlichkeit $p_\theta(y \mid x)$ ergibt:

$$p_\theta(y \mid x) = \frac{1}{Z_\theta\,(x)} \prod_{i=1}^{n+1} \exp\left(\sum_k \lambda_k \, \mathrm{f}_k(y_{i-1},\, y_i, x) + \sum_k \mu_k \, \mathrm{g}_k(\, y_i, x)\,\right) \quad (1.4)$$

C.2 CRF- Training

Um die Parameter $\theta(\lambda_1, \lambda_2, \ldots; \mu_1, \mu_2, \ldots)$ laut der Trainingsdaten zu schätzen, verwenden wir die Maximum Liklihood Estimation.

Angenommen, die Trainingsdaten bestehen aus einer Menge von N Paaren und jedes Paar hat eine Beobactungssequenz und eine entsprechende Labelsequenz $D = \left\{\left(x^{(i)}, y^{(i)}\right)\right\} \forall i = 1 \ldots N$. Dann nimmt die Log-Likelihood Funktion die Form an:

$$L(\theta) = \sum_{x,y} \mathrm{p}(\mathrm{x},\mathrm{y}) \log \mathrm{q}_\theta(y|x) \qquad (1.5)$$

Hierbei $\theta(\lambda_1, \lambda_2, \dots; \mu_1, \mu_2, \dots)$ die Parameter von Modell ist und $\mathrm{q}_\theta(y|x)$ die Bedingung für p ist**[4]**.

Die Log-Likelihood-Funktion wird sich als konkave Funktion und Kontinuität im gesamten Parameterraum beweisen. So können wir die maximale Log-Likelihood-Funktion durch die Gradientenvektormethode finden. Jedes Element im Gradientenvektor $G(\theta)$ hat einen Wert von 0 **[4]**.

$$G(\theta) = E_p[f] - E_{\mathrm{q}_\theta}[f] \qquad (1.6)$$

Derzeit gibt es viele Methoden, um das Problem der maximalen Loglikelihood-Funktion zu lösen, wie beispielsweise an Improved Iterative Scalin (IIS) und Generalized Iterative Scalin(GIS) und numerische Optimierungsmethoden wie Conjugate Gradient, quasi-Newton Methode **[4]**. Und die Standardmethode ist L-BFGS Methode (Limited Memory Variable Metrics).

L-BFGS ist eine Optimierungsmethode zweiter Ordnung, die aus der Taylor-Expansion zweiter Ordnung abgeleitet wird.[4]

$$L(\theta + \Delta) \approx L(\theta) + \Delta^T G(\theta) + \frac{1}{2}\Delta^T H(\theta)\Delta \qquad (1.7)$$

Hierbei $G(\theta)$ den Gradientenvektor ist und $H(\theta)$ die quadratische Ableitung der Log-Likelihood-Funktion $L(\theta)$ ist und auch Hessian Matrix von $L(\theta)$ gennant wird **[4]**. Wenn wir die Ableitung von (1.7) auf 0 setzen, dann kriegen wir den Wert von Inkrementieren Δ, um den Modellparameter zu aktualisieren **[3]**.

$$\Delta^{(k)} = H^{-1}(\theta^{(k)})\mathrm{G}(\theta^{(k)}) \qquad (1.8)$$

Hierbei k die Anzahl der Iterationsschritten ist. Die Hessian Matrix ist häufig sehr groß, insbesondere beim Problem der parametrischen Schätzung des CRF-Modells. Daher ist es nicht praktisch, die Inverse von einer Hessian Matrix direkt zu berechnen. Die L-BFGS Methode kann nicht direkt mit der Hessian Matrix berechnen, sondern nur die Änderung der Krümmung des Gradientenvektors im Vergleich zum vorherigen Schritt und aktualisiert sie erneut.Dann kann die Formel (1.8) umformuliert werden.

$$\Delta^{(k)} = B^k \mathrm{G}(\theta^{(k)}) \qquad (1.9)\ \textbf{[3]}$$

Hierbei B^k eine symmetrische, positiv definierte Matrix ist diese Matrix die Änderung der Krümmung durch jede Iteration des Algorithmus bezeichnet **[3]**. Da der Algorithmus einen begrenzten Speicher hat, daher zeigt der Algorithmus in jedem Iterationsschritt k, dass die zur Berechnung von B^k verwendeten Parameter separat gespeichert werden und wenn der benutzte Speicher voll ist, werden die alten Parameter gelöscht, um die neuen Parameter zu ersetzen **[6]**.

C.3 CRF Part-Of-Peech tagging

Nachdem wir das CRF-Modell aus dem Trainingsdaten gefunden haben, verwenden wir dieses Modell zum Beschriften der Beobachtungssequenzen. Dies bedeutet, dass wir eine Maximierung der Korpusverteilung zwischen der Labelsequenzen y und der Beobachtungssequenzen x machen. Die Labelsequenzen y*, die am besten die Beobachtungssequenzen x beschreiben, sind die Lösung der Gleichung:

$$y^* = argmax\{\mathrm{p(y|x)}\} \qquad (2.0)$$

Die Labelsequenzen y* kann mit dem Viterbi Algorithmus bestimmt werden **[7]**.

Wir verwenden die Formel (1.4), um $p_\theta(y^*, x)$ zu berechnen:

$$M_\mathrm{i}(y_{i-1},\ y_i \mid x) = \exp\left(\sum_k \lambda_k\ \mathrm{f}_k(y_{i-1},\ y_i, x) + \sum_k \mu_k\ \mathrm{g}_k(\ y_i, x)\ \right)$$

$$p_\theta(y \mid x) = \frac{1}{Z_\theta\,(x)} \prod_{i=1}^{n+1} M_\mathrm{i}(y_{i-1},\ y_i \mid x) \quad (2.1)$$

Da lediglich die Maximierung von $p_\theta(y|\ x)$ relevant ist, muss $Z_\theta\,(x)$ nicht berechnet werden. Die Hauptidee des Viterbi-Algorithmus besteht darin, die optimale Labelssequenz schrittweise zu erhöhen, indem Matrizen von Position 0 auf Position n abgetastet werden. Bei jedem Schritt i werden alle optimalen Sequenzen $y_1^*(y)$, die von dem Label y beendet werden, und das entsprechende Produkt $P_\mathrm{i}(y)$ geschrieben:

Schritt 1: $P_1(y_1) = M_1(y_0, y_1 \mid x)$ und $y_1^*(y_1) = y_1$.

Schritt 2: Wir machen eine Schleife von $i = 2$ bis $i = n + 1$ zum Berechnen:

$$P_\mathrm{i}(y_i) = \max P_{i-1}(y_i) \times M_\mathrm{i}(y_{i-1}, y_i \mid x)$$

$$y_i^*(y) = y_{i-1}^*(arg\max P_{i-1}(y_i) \times M_\mathrm{i}(y_{i-1}, y_i \mid x)\ .(y)$$

Wobei „.“ der Operator von Addition von String ist.

Und die Kette $y_n^*(y)$ ist die größte Wahrscheinlichkeit $p_\theta(y^* \mid x)$ und die am besten entsprechende Labelssequenz von der Beobachtungssequenz x.

3 Erkennung von Entitäten in historischen Bundestagsprotokollen

A. Beschreibung der Aufgabe

Problem

In den 18. Bundestagsprotokollen gibt es noch keine eindeutige Klassifizierung der XML-Tags. Dies führt zu vielen Schwierigkeiten, Informationen bei Abfragen zu extrahieren.

Lösung

Schritt 1: Ich verwende das CRF-Modell, um Worttypen zu kennzeichnen. Hier wird die CRF++ Bibliothek genutzt.

Schritt 2: Ich wende die Technik „Chunking“ an, um das Ergebnis der CRF++ zu verarbeiten und um Sätze zu erstellen und zu klassifizieren.

Schritt 3: Schließlich schreibe ich ein Programm and bestimme ich den geeigneten Typ des XML-Tags für jeden Satz und wird die XML-Datei exportiert.

B. Bearbeitung von Daten

Datensammlung für CRFs

Trainingsdaten zur Erstellung eines CRF-Modells werden aus den XML-Dateien der Plenasprotokolle der 18. Wahlperiode (2013 - 2017) **[8]** gesammelt.

Datenvorbereitung von Trainingsdaten

Aus den gesammelten Dateien wird der Hauptinhalt, der in dem „<TEXT>“ Tag enthalten wird, zunächst herausgefiltert. Dann wird dieser Inhalt in zwei Phasen verarbeitet, um die Genauigkeit für die Trainingsdaten sicherzustellen.

Phase 1: Ich schreibe ein Programm **[9]**, das mir automatisch helfen kann, Wörter in jedem Protokoll zu trennen, zu beschriften und Trainingsdaten zu erstellen. Da die Struktur der Daten sehr vielfältig ist, wird die Genauigkeit des Ergebnisses nach der Durchführung manchmal nicht ganz korrekt (oft mit einer Genauigkeit zwischen 85-100%) prädiktiert.

Phase 2*:* Der Zweck dieser Phase ist es, die höchste Genauigkeit der Trainingsdaten sicherzustellen. Nachdem ich die Ergebnisse des Programms in der ersten Phase erhalten habe,

werde ich das Ergebnismanuell überprüfen, um benötige Daten hinzuzufügen oder zu entfernen. Dies ist zielführend, jedoch auch ein zeitaufwendiger Schritt.

Eingabedatenstruktur

In jedem Protokoll der 18. Wahlperiode werde ich mit meinem Programm den Textinhalt des XML-Tags <TEXT> nehmen. Der Inhalt hat immer eine ähnliche Struktur (Tabelle 1).

Teil der Struktur	Zeichen der Anerkennung	Beispiel
Informationen des Plenarprotokolls	Dieser Teil befindet sich oben im Inhalt. Informationen zur Protokolle-Datei z.B. Wahlperiodenumber, Titel, Datum werden aufgeführt.	- Plenarprotokoll 18/1 Deutscher Bundestag Stenografischer Bericht. - Berlin, Dienstag, den 22. Oktober 2013
Vorspann	Beginnt mit der Zeile „I n h a l t :“.	I n h a l t :
Sitzungsbeginn	Beginnt mit der Zeile „Beginn: <*Zeit*> <*Zeiteinheit*>“.	Beginn: 11.00 Uhr
Tagesordnungspunkt 1	Enhälte „Tagesordnungspunkt(e/en/em)“ oder „Zusatzpunkt(e/en/em)“ und wird mit einem Kolon geendet.	- Ich rufe den Tagesordnungspunkt 1 auf: - Wir kommen damit zu Tagesordnungspunkt 1 mit dem gerade vereinbarten Zusatzpunkt 1:
Tagesordnungspunkt 2		
.....		
Tagesordnungspunkt n		
Sitzungsende	Die Zeile „(Schluss: <*Zeit*> <*Zeiteinheit*>)“.	(Schluss: 21.28 Uhr)
Anlegen und Rednerliste	Folgt der Zeile des Sitzungsendes	

Tabelle 1: Struktur von dem <TEXT>-Tag aller XML–Dateien der Protokolle der 18. Wahlperiode.

Trainingsdaten von CRF Modell

Daneben enthält der Inhalt von <TEXT>-Tag auch viele Worttypen und Satztypen mit unterschiedlichen Strukturen. Die Aufgabe ist es, mit CRF-Modell den richtigen Typ für jedes Wort und jeden Satz zu identifizieren. Hierbei ist das Erzielen eines optimalen Ergebnisses schwierig, denn je mehr Worttypen es gibt, desto leichter ist ein falsches CRF zu bestimmen. Um diese Komplexität zu minimieren, werde ich hier versuchen, so viele Typen von Wörtern wie möglich zur Verwendung als Worterkennungsbedingungen aufzulisten. Es werden separate Bedingungen

für eine Wortgruppe oder für ein Wort angegeben, sodass das CRFs die Wörter leichter erkennen kann und das Tagging von Wörtern präziser und schneller durchgeführt wird. Die folgende Tabelle zeigt die Liste der Worttypen und ihrer Bezeichnungen.

Worttyp	**Bedeutung**	**Tag**	**Beispiel**
BE	Beginn	BEGIN	Beginn
CONTENT	Inhalt	CONTENT	I n h a l t
COL	Kolon	COLON	:
NUM	Nummer	NUMBER	0,1,2,3....usw
DOT	Punkt, Komma, Ausrufezeichen,	DOT	. ! ,
QUES	Fragezeichen	QUESTION	?
STATE	Bundesland	STATE	Bayern, Berlin,...usw
BRACKET	Klammer	BBRACKET, EBRACKET	(,)
INQUEST	Zwischenfrage	INTERQUESTION	
TITLE	Titel	TITLE	Dr. Prof.
NAME	Vorname und Nachname	NAME	Sebastian Brehm
AGEN	Tagesordnungspunkt und Zusatzpunkt	AGENDA	Tagesordnungspunkt order Zusatzpunkt
PLE	Plenarprotokoll	PLENARYPRO	Plenarprotokoll
DIV	Schrägstrich	DIV	/
MEETING	Sitzung	MEETING	Sitzung
PRE	Präsident, Vizepräsident, Alterspräsident	PRESIDENT	Präsident, Vizepräsident, Alterspräsident
FRACT	Fraktion	FRACTION	CDU, CSU, SPD,..usw
WEEKDAY	Wochentag	WEEKDAY	Montag, Dienstag,..usw
MONTH	Monat	MONTH	Januar, Februar,...usw
TIME	Zeiteinheit	TIME	Uhr, Monat, Jahr,...usw
MOOD	Stimmung	MOOD	Beifall, Heiterkeit,..usw
END	Schluss	END	Schluss
PAR	Paragraph	PARAGRAPH	in anderen Fällen

Tabelle 2: Liste der Worttypen und ihrer Bezeichnungen in einem Protokoll.

Da IOB2 (Inside, Outside, Beginning) das Eingabeformat von dem CRF-Modell ist, müssen die gesammelten Daten in die IOB2-Struktur konvertiert werden.

(	BRACKET	NoCap	**B-BBRACKET**
Dr.	TITLE	Cap	**B-TITLE**
Diether	NAME	Cap	**B-NAME**
Dehm	NAME	Cap	**I-NAME**
[	PAR	NoCap	**B-PARAGRAPH**
DIE	PAR	Cap	**B-FRACTION**
LINKE	FRACT	Cap	**I-FRACTION**
]	PAR	NoCap	**B-PARAGRAPH**
:	COL	NoCap	**B-COLON**
Hallo	PAR	Cap	**B-PARAGRAPH**
Leute	PAR	Cap	**I-PARAGRAPH**
!	DOT	NoCap	**B-DOT**
)	BRACKET	NoCap	**B-EBRACKET**

Abbildung 3: Beispiel für einen Satz "Kommentar".

Die Abbildung 3 zeigt:

- Die erste Spalte ist ein Wort der Beobachtungsdaten.
- Die zweite Spalte ist der Worttyp
- Die Spalte 3 zeigt, ob das Wort ein Großbuchstabe (Cap) ist oder nicht (NoCap).
- Die Spalte 4 ist der Tag des Wortes im IOB2-Format.

Hier werden die Spalten 1, 2 und 3 als Eingabewerte für das CRFs-Modell betrachtet, sodass es beim Testen den Tag des Wortes (in der Spalte 4) prädiktieren kann.

Musterdatei der CRF++

Mit der CRF++ **[10]** ist es einfach, die Verbindungen zwischen den Bedingungen von Trainingsdaten durch einer Musterdatei zu erstellen. Anhand der Musterdatei lässt sich die Logik sowie das Tagging der CRF++ überprüfen, deswegen spielt diese Datei eine wichtige Rolle. Und wenn wir die Musterdatei falsch schreiben, wird CRF++ nicht richtig funktionieren und dann ein falsches Ergebnis geben.

```
# Unigram

U00:%x[0,0]/%x[1,0]/%x[2,0]
U01:%x[-1,0]
U02:%x[0,0]
U03:%x[1,0]
U04:%x[0,0]/%x[0,1]/%x[1,1]/%x[0,2]
U05:%x[-1,1]/%x[0,1]/%x[1,1]/%x[2,1]

# Bigram
B
```

Abbildung 4: Der Inhalt von meiner Musterdatei.

Hierbei x ein zweidimensionales Array ist, das die Zeilen- und Spalteninformationen der Trainingsdaten enthält. Und x[i, j] ist der Wert des Elementes in der Zeile i und der Spalte j.

Zum Beispiel haben wir Trainingsdaten wie folgt:

Dr.	TITLE	Cap	**B-TITLE**
Diether	NAME	Cap	**B-NAME**
Dehm	NAME	Cap	**I-NAME**

Damit haben wir x = [[Dr. , TITLE, Cap], [Diether , NAME, Cap], [Dehm , NAME, Cap]].

Hier gibt es zwei verschiedene Arten, die sich leicht durch den ersten Buchstaben jeder Zeile erkennen lassen. Der erste Buchstabe 'B' zeigt, dass diese Zeile ein Bigram-Muster ist, wodurch die Ergebnisse der Wortbeschiftung automatisch zusammengeführt werden. Der erste Buchstabe "U" zeigt, dass diese ein Unigramm-Muster ist. Wenn wir ein Unigram-Muster (z.B. "U02:% x [0, 0]" verarbeiten, dann erstellt die CRF++ automatisch Funktionen, wie folgend aufgeführt:

func1 = if (output = B-TITLE and feature = „U02:Dr.“) return 1 else return 0
func2 = if (output = B-NAME and feature = „U02:Diether“) return 1 else return 0
func3 = if (output = I-NAME and feature = „U02:Dehm“) return 1 else return 0

Anmerkung:

Die Anzahl der Funktionen entspricht der Anzahl der Beschriftungen, multipiliziert mit den Anzahl an Orientierungszeichenfolgen.

CRF Modelldatei erstellen

Nach dem Erstellen der Musterdatei, wird CRF++ zum Erstellen einer weiteren Modelldatei für Testdaten genutzt.

Syntax:

```
crf_learn template_file train_file model_file
```

Eingabedaten von CRF++ bei Testing

Die Testdaten sind in 3 Spalten gegliedert: Beobachtungsdaten, Worttypen und Großbuchstaben.

Ich	PAR	Cap
rufe	PAR	NoCap
die	PAR	NoCap
Tagesordnungspunkt	AGEN	Cap
6	NUM	NoCap
auf	PAR	NoCap
:	COL	NoCap

Abbildung 5: Beispiel von einer Testdatei.

Testdurchführung

Wir werden die generierte Modelldatei verwenden, um die Wörter für die Testdaten zu beschriften.

Syntax:

```
crf_test -m model_file test_files
```

Ich	PAR	Cap	**B-PARAGRAPH**
rufe	PAR	NoCap	**I-PARAGRAPH**
die	PAR	NoCap	**I-PARAGRAPH**
Tagesordnungspunkt	AGEN	Cap	**B-AGENDA**
6	NUM	NoCap	**B-NUMBER**
auf	PAR	NoCap	**I-PARAGRAPH**
:	COL	NoCap	**B-COLON**

Abbildung 6: Ergebnis des vorherigen Beispiels aus der Abbildung 5.

C. Chunking

Chunking ist eine Technik und eine Prozedur zur Verbesserung der naturalistischen Datenanalyse. In der Informatik steht Chunking für das Einteilen von Entitäten in Gruppen. Eine Gruppe besteht dabei immer aus „Chunk“ Informationseinheiten, die gemeinsame Merkmalen besitzen.

Eingabe von Chunking

Durch CRF haben wir alle Wörter im IOB2-Format beschriftet. Jetzt werden wir die Chunk-Bibliothek von NLTK verwenden, damit die IOB2-Tags zu Chunks kombiniert werden können.

Label	**Tag**
Hallo	B-PARAGRAPH
Leute	I-PARAGRAPH
Ich	B-PARAGRAPH
Rufe	I-PARAGRAPH
Die	I-PARAGRAPH
Tagesordnungspunkt	B-AGENDA
6	B-NUMBER
Auf	B-PARAGRAPH
:	B-COLON

Ergebnis durch CRFs

Label	**Tag**
Hallo Leute	PARAGRAPH
Ich rufe die	PARAGRAPH
Tagesordnungspunkt	AGENDA
6	NUMBER
Auf	PARAGRAPH
:	COLON

Ergebnis durch Chunking.

Abbildung 7: Beispielhaftes Ergebnis des Chunkings mit vorheriger Tag-Bestimmung durch das CRFs.

Das Ergebnis von Chunking wird in einem Array gespeichert. Jedes Chunk-Element von dem Array ist ein Tupel mit zwei Werten: Label und Tag des Labels (Abbildung 7).

```
chunk_data = [
                ("Hallo Leute", PARAGRAPH),
                ("Ich rufe die", PARAGRAPH),
                ("Tagesordnungspunkt", AGENDA),
                ("NUMBER", 6),
                ("auf", PARAGRAPH),
                (":", COLON)
            ]
```

Abbildung 8: Ergebnis von Chunking wird in einem Array gespeichert.

Chunking-Regeln

Als nächstes verwenden wir Chunking-Regeln, um die Satzstruktur zu definieren, die Tupel zu kombinieren und Sätze zu bilden. Die folgende Tabelle zeigt die Satztypen und ihre Chunking-Regel.

Index	Satztyp	Chunking Regeln	Beispiel
1	Plenarprotokoll	<PLENARYPRO><NUMBER><DIV><NUMBER><PARAGRAPH>	Plenarprotokoll 18/5 Deutscher Bundestag Stenografischer Bericht
2	Sitzung	<NUMBER><DOT><MEETING>	5. Sitzung
3	Inhalt	<CONTENT>	I n h a l t :
4	Sitzungsbeginn	<BEGIN><COLON><NUMBER><DOT><NUMBER><TIME>	Beginn: 9:00 Uhr
5	Tag	<STATE><DOT><WEEKDAY><DOT><PARAGRAPH>?<NUMBER><DOT>?<MONTH>	Berlin, Donnerstag, den 28. November 2013
6	Präsident	Der Satz beginnt mit dem <PRESIDENT> und endet mit dem <COLON>	Präsident Dr. Norbert Lammert:
7	Name	Der Satz beginnt mit dem <TITLE>?<NAME> und endet mit dem <COLON>	Dr. Angela Merkel, Bundeskanzlerin:

8	Tagesordnungspunkt	Der Satz enthält <AGENDA><NUMBER> und endet mit dem <COLON>	Ich rufe die Tagesordnungspunkt 1 auf:
9	Sitzungsende	<BBRACHKET><END><COLON><NUMBER><DOT><NUMBER><TIME><EBRACHKET>	(Schluss: 12.40 Uhr)
10	Kommentar	Der Satz beginnt mit dem <BBRACKET> und endet mit dem <EBRACKET>.	(Beifall bei der CDU/CSU und der SPD)
11	Paragraph	in anderen Fällen	Guten Morgen!

Tabelle 3: Übersicht von Satztypen und ihrer Chunking-Regeln.

Diese Regeln werden einer Liste zugefügt, welche als Eingabe für das Objekt nltk.RegexpParser genutzt wird, um die Elemente des Chunking-Resultats (wie in Abbildung 8 zu sehen) in Gruppen zu kombinieren und schließlich entsprechend jeder Gruppe Sätze zu bilden.

Anmerkung:

Wenn mehrere Regeln für den Inhalt greifen, verwendet das RegexpParser-Objekt nur die Regel, die den kleinsten Index-Wert (siehe Tabelle 3) hat. So zum Beispiel hat der Satztyp "PARAGRAPH" die geringste Priority.

Hier kommt es bei der Gruppierung zu einem Problem: Entnommen der resultierenden Chunking-Liste aus Abbildung 7, lässt sich feststellen, dass es dort zwei verschiedenen Satztypen („Hello Leute“ und “Ich rufe die Tagesordnungspunkt 6 auf“) gibt. Der Algorithmus erkennt nicht, wo der Anfang und das Ende eines Satzes zind. Dadurch wird die Satzbildung komplizierter, fehleranfälliger und zeitaufwändiger bei Chunking, weil das RefexpParser auch nicht wissen kann, welchen Regeln es anwenden soll, um die vernünftigsten Ergebnisse zu erzielen.

Wie in der Abildung 7 haben wir 2 Sätze "Hallo Leute" und "Ich rufe Tagesordnungspunkt 6 auf:". Wir wissen, dass die Kette „Hallo Leute“ den Tag von Paragraph hat und das RefexpParser es vorziehen wird, zuerst den Regeln „Tagesordnungspunkt“ im Vergleich mit den „Paragraph“ zu verwenden, da die Regel „Paragraph“ einen geringeren Indexwert hat. Desweiteren greifen beim Tagesordnungspunkt die Regel „Der Satz enhält <AGENDA><NUMBER> und schließlich <COLON>“. Hieraus ergibt sich, dass die Regel egal ist, welche Tags vor <AGENDA><NUMBER> stehen, sodass „Hallo Leute“ als separater Teil abseits von „Tagesordnungspunkt“ betrachtet wird (Abbildung 9).

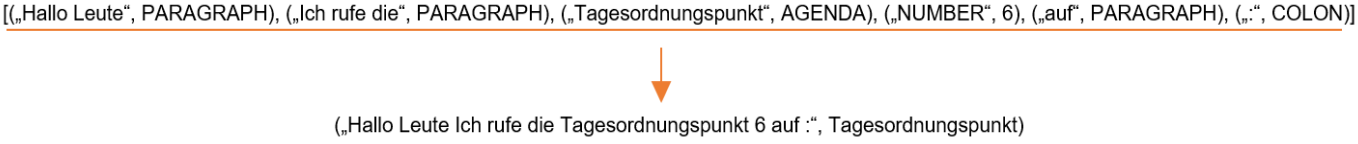

Abbildung 9: Problem bei dem Gruppierungsprozess zur Satzbildung.

Um dieses Problem zu lösen, füge ich vor jedem Satz einen Tag <START-LINE> und am Ende jedes Satzes einen Tag <END-LINE> ein. Dies macht den Chunking-Prozess einfacher, schneller und fehlerfrei (Abbildung 10).

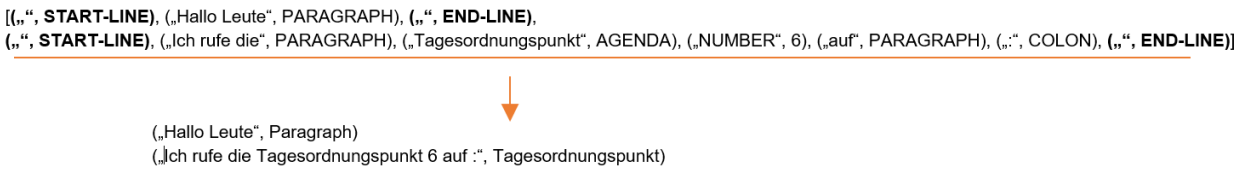

Abbildung 10: Die Lösung für das Problem bei der Gruppierung zur Satzbildung.

Ergebnis von Chunking

Durch Chunking-Regeln erhalten wir eine Liste von baumartigen Sätzen. Jeder Baum enthält den Namen des Satztyps und ein Array mit einer Liste der zusammengesetzten Chunks:

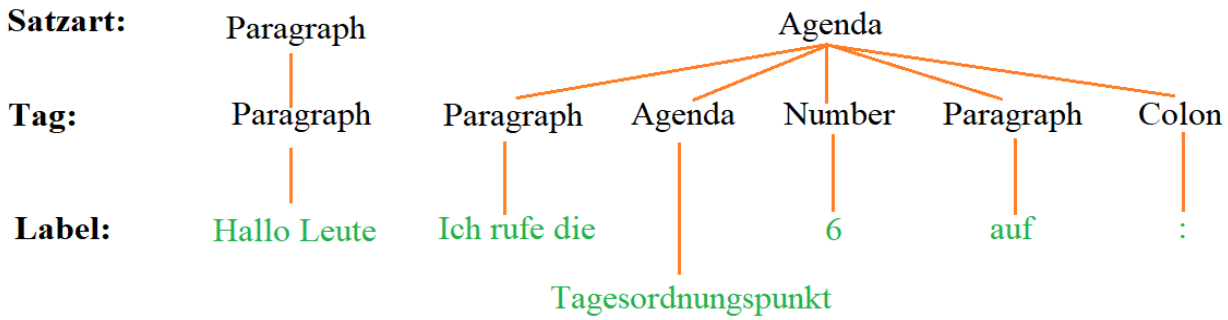

Abbildung 11: Beispielhaftes Chunking-Ergebnis in Form einer Baumstruktur.

Die baumartigen Daten von der Abbildung 11 werden als folgende Daten gespeichert:

<< **Satztyp** >> (<<**Label_1**>> / <<**Tag_1**>> ... <<**Label_n**>> / <<**Tag_n**>>)

Beispiel:

Paragraph (Hallo Leute/***Paragraph***)

Agenda(Ich rufe die/**Paragraph** Tagesordnungspunkt/**Agenda** 6/**Number** auf/**Paragraph** :/**Colon**)

D. Export von XML-Datei

Layout von XML-Datei

Basierend auf dem Layout der XML-Datei der 19. Wahlperiode konnte ich das Layout für die XML-Datei von Protokolle 18 bestimmen, wie folglich zu sehen:

Encoding: UTF-8 Stylesheet: dbtplenarprotokoll.css Doctype: <!DOCTYPE dbtplenarprotokoll SYSTEM "dbtplenarprotokoll.dtd">				
Name				**Properties**
Dbtplenar-protokoll				sitzung-ort
				herausgeber
				wahlperiode
				sitzung-nr
				sitzung-datum
Vorspann				
	kopfdaten			
		plenarprotokoll-nummer		
			wahlperiode	
			sitzungsnr	
		herausgeber		
		berichtart		
		sitzungstitel		
			sitzungsnr	
		veranstaltungsdaten		
			ort	
			datum	
	inhaltsverzeichnis			
		ivz-titel		
		ivz-eintrag		
		ivz-block		
sitzungsverlauf				

		sitzungsbeginn			sitzung-start-uhrzeit
			Inhalt-tag (Abbildung 12)		
		tagesordnungspunkt			top-id
			Inhalt-tag		
		sitzungsende			sitzung-ende-uhrzeit
	anlagen				
		anlagen-titel			
		anlage			
			anlagen-titel		
			anlagen-text		
	rednerliste				
		Redner			
			name		
				titel	
				vorname	
				nachname	
				fraktion	

Abbildung 12: Layout von XML-Datei der 18. Wahlperiode.

*__Inhalt-tag__

Name	**Properties**
p	klasse
kommentar	
rede	id
p	Klasse="redner"
redner	id
name	
titel	
vorname	
nachname	
fraktion	
rolle	
Inhalt-tag	
präsident	name
titel	
vorname	
nachname	
rolle	
Inhalt-tag	
zwischenfrage	rede
präsident	

Abbildung 13: Layout der Liste „Inhalt-tag".

Als Nächstes verwenden wir die Bibliothek xml.etree.ElementTree, um XML-Tags zu erstellen. Der Implementierungsprozess ist wie folgt zusammengefasst:

Schritt 1: Lese jeden Baum in dem Ergebnis von Chunking. (Abbildung 10)

Schritt 2: Basierend auf dem Namen des Satztyps bestimme ich das entsprechende XML-Tag.

Schritt 3: Verwende die Bibliothek ElementTree, um XML-Tags zu erstellen.

Schritt 4: Wiederhole die Schritte 2 und 3 bis zum Ende von Liste des Baums. Wenn alle Bäume schon durchgeführt werden, kommen wir zum Schritt 5.

Schritt 5: XML-Datei schreiben.

E. Durchführung des Programms und Testen

Mein Programm wurdet auf Github **[10]** hochgeladen. Dort kann man sehen, wie das Programm funktioniert und wie das Ergebnis aussieht.

4 Fazit

Die Arbeit zeigt eine Einführung des CRF-Modells und damit auch, dass die XML Dateien der Plenarprotokolle der 18. Wahlperiode mit dem CRFs-Modell beschriften werden können. Anschließend werden diese mithilfe der Chunking Technik erfolgreich zu Tags gruppiert, sodass Sätze gebildet und XML-Dateien erstellt werden können. Im Prozess dieser Forschungsarbeit wurde deutlich, dass noch einige Risiken auftreten können, die dazu führen, dass das CRFs-Modell nicht korrekt funktioniert und falsche Ergebnisse liefert:

- Da die Daten der Protokolle sehr groß sind und unterschiedliche Struktur aufweisen, muss die Datenaufbereitung für das CRF-Modell extensiv betrieben werden, damit das CRF-Modell alle Wörter von den verschiedenen Datensätzen beschriften kann. In dieser Arbeit war es lediglich möglich, diese Daten manuell zu überprüfen, weshalb nur ein kleiner Datensatz vorbereitet werden konnte und restliche Informationen bei der finalen Analyse fehlen könnten.

- Der Inhalt der CRF++ Musterdatei (Template file) ist der entscheidende Faktor für das Ergebnis des Modells. Zurzeit werden nur drei Bedingungen (Wort, Typ, Großbuchstaben) aufgelistet, um den Worttyp eines Wortes zu definieren. Wegen der vielfältigen und strukturierten Daten wären die oben genannten 3 Bedingungen wahrscheinlich noch nicht ausreichend.

- Da durch Leerzeilen in den Dateien viele Paradoxe in den Protokollen vorliegen, führt dies zu Schwierigkeiten bei der Verarbeitung der Eingabe und dem Chunking. Obwohl viel Zeit für die Optimierung des Programms aufgebracht wurde, um dieses Problem zu beheben, bleiben Teile der Dateien wahrscheinlich noch nicht richtig vorbereitet.

5 Ausblick

Weitere Forschungsrichtungen

- Fehlerprüfung und Verbesserung der Trainingsdaten für das CRF-Modell.
- Genauigkeit und Laufzeit prüfen.
- Einige fehlende XML-Tags und Attribute hinzufügen.
- Mit der Tkinter-Bibliothek eine Software als Windows Form Application erstellen.

Literatur

[1] A. McCallum, D. Freitag, and F. Pereia. Maximum entropy markov models for information extraction and segmentation. In Proc. Interational Conference on Machine Learning, 2000.

[2] J. Lafferty, A. McCallum, and F. Pereia. Conditional ramdom fields: probabilistic models for segmenting and labeling sequence data. In International Conference on Machine Learning, 2001.

[3] Hammersley, J., & Clifford. P. Markov fields on finite graphs and lattices. Unpublished manuscript ,1971.

[4] Malouf, Robert. (2002). A Comparison of Algorithms for Maximum Entropy Parameter Estimation. Proceedings of the Sixth Conference on Natural Language Learning. 20. 49-55. 10.3115/1118853.1118871.

[5] Huang, Han-Shen & Chang, Yu-Ming & Hsu, Chun-Nan. (2007). Training Conditional Random Fields by Periodic Step Size Adaptation for Large-Scale Text Mining. 511 - 516. 10.1109/ICDM.2007.39.

[6] Dong C.Liu & Jorge Nocedal. On the limited memory BFGS method for large scale optimization. Mathematical Programming 45 (1989), pp 503-528.

[7] Klinger, R. & Tomanek, K.. (2007). Classical Probabilistic Models and Conditional Random Fields. Tech Rep TR07-2-013.

[8] Offene Daten auf einer offenen Festplatte, Deutscher Bundestag. Online verfügbar unter: https://www.bundestag.de/services/opendata, letzter Abruf 30.01.2021.

[9] Truong An Nguyen, CRF Entity detection in German federal parliament. Online verfügbar unter: https://github.com/yiimnta/CRF-Entity-detection-in-German-federal-parliament, letzter Abruf 30.01.2021.

[10] CRF++: Yet Another CRF Tookit. Online verfügbar unter: https://github.com/yiimnta/CRF-Entity-detection-in-German-federal-parliament, letzter Abruf 30.01.2021.