

# Hidden Markov Model als POS-Tagger für kleine, bedrohte Sprachen (Urum)

Aleksandr Schamberger

Humboldt-Universität zu Berlin  
Institut für Slawistik und Hungarologie  
Einführung in die Computerlinguistik (mit Anwendung auf slawische Sprachen)  
WS 2023/24

10. Februar 2024

# Gliederung

- 1 Was sind DoReCo und Urum?
  - DoReCo
  - Urum
- 2 Was ist ein Hidden Markov Model (HMM)?
- 3 HMM als POS-Tagger
- 4 Meine Implementierung eines HMM als POS-Tagger für Urum

# Gliederung

1 Was sind DoReCo und Urum?

- DoReCo
- Urum

2 Was ist ein Hidden Markov Model (HMM)?

3 HMM als POS-Tagger

4 Meine Implementierung eines HMM als POS-Tagger für Urum

# DoReCo: Language Documentation Reference Corpus

# DoReCo

Language Documentation Reference Corpus



- Webseite des Projekts: <https://doreco.info/>
- Webseite des Repositoriums: <https://doreco.huma-num.fr/>

# DoReCo: Language Documentation Reference Corpus

- DoReCo: Language **D**ocumentation **R**eference **C**orpus

# DoReCo: Language Documentation Reference Corpus

- DoReCo: Language **D**ocumentation **R**eference **C**orpus
- weltweit 51 kleine, bedrohte Sprachen; ursprünglich gesammelt in der Feldforschung

# DoReCo: Language Documentation Reference Corpus

- DoReCo: Language **D**ocumentation **R**eference **C**orpus
- weltweit 51 kleine, bedrohte Sprachen; ursprünglich gesammelt in der Feldforschung
- Gesprochene Daten (mit Audioaufnahmen) meist erzählerischer (narrativer) Texte

# DoReCo: Language Documentation Reference Corpus

- DoReCo: Language **D**ocumentation **R**eference **C**orpus
- weltweit 51 kleine, bedrohte Sprachen; ursprünglich gesammelt in der Feldforschung
- Gesprochene Daten (mit Audioaufnahmen) meist erzählerischer (narrativer) Texte
- Transkriptionen sind auf der Phon-Ebene **zeitaligniert**.

# DoReCo: Language Documentation Reference Corpus

- DoReCo: Language **D**ocumentation **R**eference **C**orpus
- weltweit 51 kleine, bedrohte Sprachen; ursprünglich gesammelt in der Feldforschung
- Gesprochene Daten (mit Audioaufnahmen) meist erzählerischer (narrativer) Texte
- Transkriptionen sind auf der Phon-Ebene **zeitaligniert**.
- 38 der 51 Sprachen verfügen über morphologische, zeitalignierte Annotationen

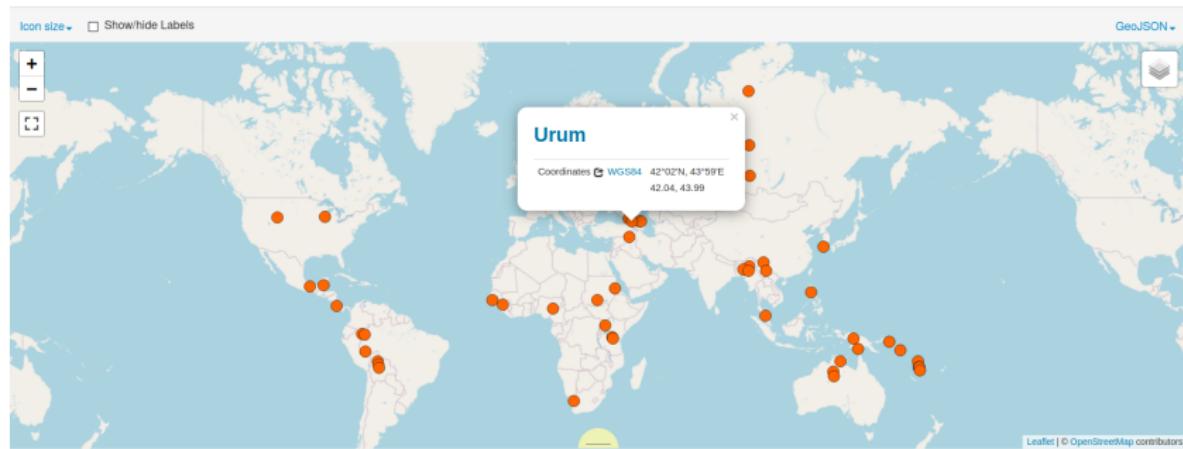
# DoReCo: Language Documentation Reference Corpus

- DoReCo: Language **D**ocumentation **R**eference **C**orpus
- weltweit 51 kleine, bedrohte Sprachen; ursprünglich gesammelt in der Feldforschung
- Gesprochene Daten (mit Audioaufnahmen) meist erzählerischer (narrativer) Texte
- Transkriptionen sind auf der Phon-Ebene **zeitaligniert**.
- 38 der 51 Sprachen verfügen über morphologische, zeitalignierte Annotationen
- Korpusdaten (Transkriptionen/Annotationen) als eaf- (ELAN), TextGrid- und csv-Dateien

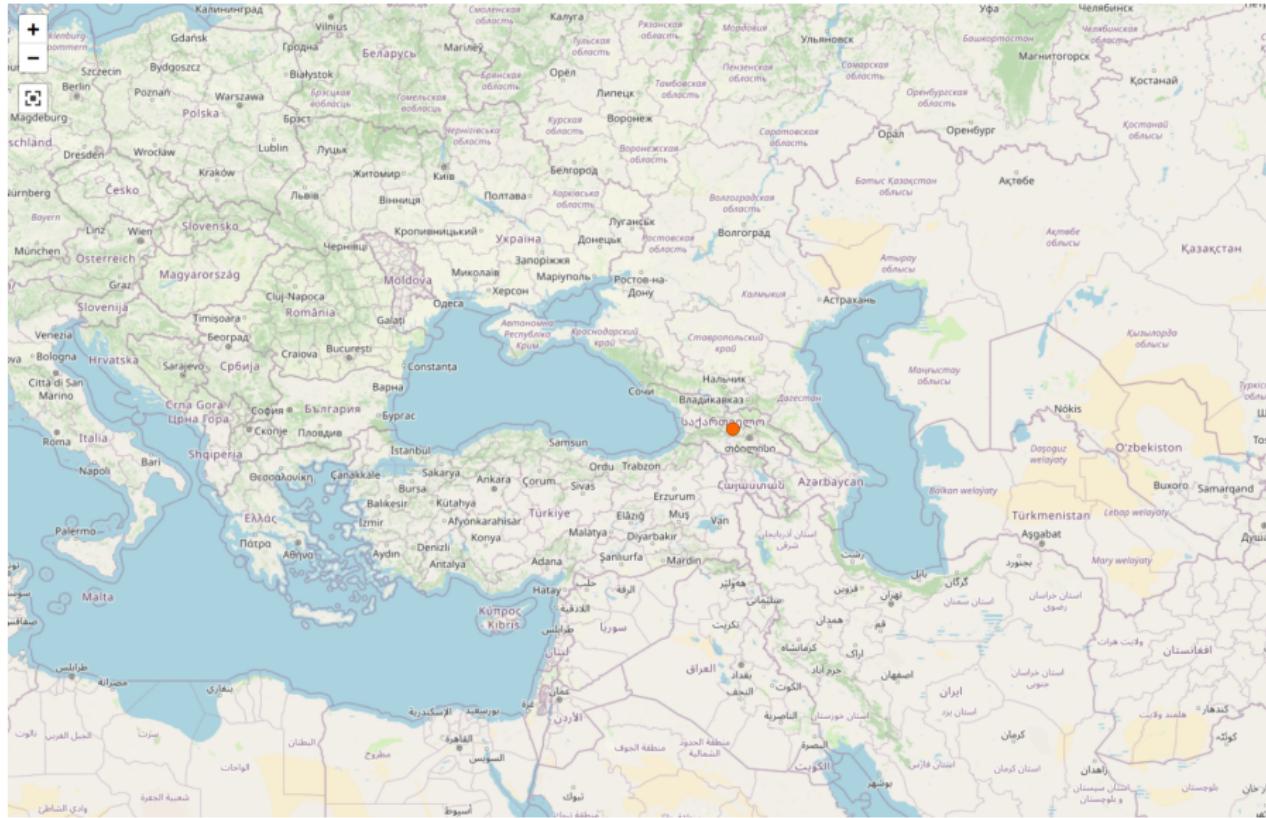
# DoReCo: Beispiel in ELAN

ref@A01 [22]	0001_DoReCo_doreco_urum1249_UUM-TXT-AN-00000-A01					
ft@A01 [22]	To my mind, they came two hundred years ago from Trabzon.					
tx@A01 [22]	bänä ğala gäldilär iki yuz el iräli trapezondadan					
wd@A01 [90]	bänä	ğala	gäldilär		< p:>	
mb@A01 [143]	bän	-ä	ğal	-a	gäl	-dī -lär
gl@A01 [143]	1.SG	-D	stay	-GER	come	-PS -PL
ps@A01 [143]	PN	-c	V	-tam	V	-ta -num
ph@A01 [342]	b {	n	{ G	a 5 a	}\	{   d  5   {   r \
doreco-mb-al [143]	****	**	*****	****	*****	*** ****
ge-a@unknown [22]	1.SG -DAT stay -GER come -PST -PL two hundred year before Trabzon-LOC-ABL					

# Urum: DoReCo Karte



# Urum: Karte: Genauer



# Urum

- Kaukasisches Urum (nicht *Crimean Urum*)

# Urum

- Kaukasisches Urum (nicht *Crimean Urum*)
- Türkische Sprache gesprochen in Georgien (in den Hochlanden von K'vemo K'art'li) u.a. am Fluss Tsalka

# Urum

- Kaukasisches Urum (nicht *Crimean Urum*)
- Türkische Sprache gesprochen in Georgien (in den Hochlanden von K'vemo K'art'li) u.a. am Fluss Tsalka
- Türkisch sprechende Pontogriechen, die Anatolien zu Beginn des 19. Jahrhundert verließen

# Urum

- Kaukasisches Urum (nicht *Crimean Urum*)
- Türkische Sprache gesprochen in Georgien (in den Hochlanden von K'vemo K'art'li) u.a. am Fluss Tsalka
- Türkisch sprechende Pontogriechen, die Anatolien zu Beginn des 19. Jahrhundert verließen
- Menschen mit griechischer Ethnizität in der Region Tsalka (die meisten sprechen Urum):

# Urum

- Kaukasisches Urum (nicht *Crimean Urum*)
- Türkische Sprache gesprochen in Georgien (in den Hochlanden von K'vemo K'art'li) u.a. am Fluss Tsalka
- Türkisch sprechende Pontogriechen, die Anatolien zu Beginn des 19. Jahrhundert verließen
- Menschen mit griechischer Ethnizität in der Region Tsalka (die meisten sprechen Urum):
  - 1979: 30.811

# Urum

- Kaukasisches Urum (nicht *Crimean Urum*)
- Türkische Sprache gesprochen in Georgien (in den Hochlanden von K'vemo K'art'li) u.a. am Fluss Tsalka
- Türkisch sprechende Pontogriechen, die Anatolien zu Beginn des 19. Jahrhundert verließen
- Menschen mit griechischer Ethnizität in der Region Tsalka (die meisten sprechen Urum):
  - 1979: 30.811
  - 2002: 4.589

# Urum

- Kaukasisches Urum (nicht *Crimean Urum*)
- Türkische Sprache gesprochen in Georgien (in den Hochlanden von K'vemo K'art'li) u.a. am Fluss Tsalka
- Türkisch sprechende Pontogriechen, die Anatolien zu Beginn des 19. Jahrhundert verließen
- Menschen mit griechischer Ethnizität in der Region Tsalka (die meisten sprechen Urum):
  - 1979: 30.811
  - 2002: 4.589
  - 2005: 1.500

# Urum

- Kaukasisches Urum (nicht *Crimean Urum*)
- Türkische Sprache gesprochen in Georgien (in den Hochlanden von K'vemo K'art'li) u.a. am Fluss Tsalka
- Türkisch sprechende Pontogriechen, die Anatolien zu Beginn des 19. Jahrhundert verließen
- Menschen mit griechischer Ethnizität in der Region Tsalka (die meisten sprechen Urum):
  - 1979: 30.811
  - 2002: 4.589
  - 2005: 1.500
- Bilingual russisch (Urum enthält viele russische Lehnwörter bzw. Code Switching findet oft statt)

# Warum Urum?

- NLP-Anwendung auf/für kleine, bedrohte Sprachen, was sonst selten bzw. (meiner Kenntnis nach) nicht existent ist.

# Warum Urum?

- NLP-Anwendung auf/für kleine, bedrohte Sprachen, was sonst selten bzw. (meiner Kenntnis nach) nicht existent ist.
- Verwandt mit dem Türkischen; Vergleich mit Sprachmodellen des Türkischen

# Warum Urum?

- NLP-Anwendung auf/für kleine, bedrohte Sprachen, was sonst selten bzw. (meiner Kenntnis nach) nicht existent ist.
- Verwandt mit dem Türkischen; Vergleich mit Sprachmodellen des Türkischen
- Kenntnisse für die Sprachdaten, aber auch die Sprache und ihre Struktur selbst erlangen.

# Warum Urum?

- NLP-Anwendung auf/für kleine, bedrohte Sprachen, was sonst selten bzw. (meiner Kenntnis nach) nicht existent ist.
- Verwandt mit dem Türkischen; Vergleich mit Sprachmodellen des Türkischen
- Kenntnisse für die Sprachdaten, aber auch die Sprache und ihre Struktur selbst erlangen.
- Mit NLP-Lösungen anderer, großer Sprachen vergleichen.

# Gliederung

1 Was sind DoReCo und Urum?

- DoReCo
- Urum

2 Was ist ein Hidden Markov Model (HMM)?

3 HMM als POS-Tagger

4 Meine Implementierung eines HMM als POS-Tagger für Urum

# Was ist ein Hidden Markov Model (HMM)?

- H = Hidden

- M = Markov

- M = Model

# Was ist ein Hidden Markov Model (HMM)?

- H = Hidden
- M = Markov
- M = Model
  - Stochastisches Modell zur Vorhersage von Zustandsabfolgen (sogenannten **Markovketten**). Bspw. Wetter, ob der Bus kommt.

# Was ist ein Hidden Markov Model (HMM)?

- H = Hidden
  
- M = Markov
  - Andrei Andrijewitsch Markow, russischer Mathematiker (1856-1922); Vorhersage von Zuständen basiert auf folgender Annahme:
  
- M = Model
  - Stochastisches Modell zur Vorhersage von Zustandsabfolgen (sogenannten **Markovketten**). Bspw. Wetter, ob der Bus kommt.

# Was ist ein Hidden Markov Model (HMM)?

- H = Hidden
  
- M = Markov
  - Andrei Andrijewitsch Markow, russischer Mathematiker (1856-1922); Vorhersage von Zuständen basiert auf folgender Annahme:
  - **Markov Annahme:**  $P(q_i = a | q_i \dots q_{i-1}) = P(q_i = a | q_{i-1})$
  
- M = Model
  - Stochastisches Modell zur Vorhersage von Zustandsabfolgen (sogenannten **Markovketten**). Bspw. Wetter, ob der Bus kommt.

# Was ist ein Hidden Markov Model (HMM)?

- H = Hidden
  
- M = Markov
  - Andrei Andrijewitsch Markow, russischer Mathematiker (1856-1922); Vorhersage von Zuständen basiert auf folgender Annahme:
  - **Markov Annahme:**  $P(q_i = a | q_1 \dots q_{i-1}) = P(q_i = a | q_{i-1})$   
Die Wahrscheinlichkeit eines Zustandes  $q_i$  innerhalb einer Kette vorheriger Zustände  $q_1 \dots q_{i-1}$  wird einzig auf Basis der Wahrscheinlichkeit des vorherigen Zustandes  $q_{i-1}$  berechnet.
- M = Model
  - Stochastisches Modell zur Vorhersage von Zustandsabfolgen (sogenannten **Markovketten**). Bspw. Wetter, ob der Bus kommt.

# Was ist ein Hidden Markov Model (HMM)?

- H = Hidden
  - Die betrachteten Zustände sind versteckt bzw. nicht direkt beobachtbar.
- M = Markov
  - Andrei Andrijewitsch Markow, russischer Mathematiker (1856-1922); Vorhersage von Zuständen basiert auf folgender Annahme:
    - **Markov Annahme:**  $P(q_i = a | q_1 \dots q_{i-1}) = P(q_i = a | q_{i-1})$   
Die Wahrscheinlichkeit eines Zustandes  $q_i$  innerhalb einer Kette vorheriger Zustände  $q_1 \dots q_{i-1}$  wird einzig auf Basis der Wahrscheinlichkeit des vorherigen Zustandes  $q_{i-1}$  berechnet.
- M = Model
  - Stochastisches Modell zur Vorhersage von Zustandsabfolgen (sogenannten **Markovketten**). Bspw. Wetter, ob der Bus kommt.

# Hidden Markov Model

$Q = q_1 q_2 \dots q_N$	a set of $N$ <b>states</b>
$A = a_{11} \dots a_{ij} \dots a_{NN}$	a <b>transition probability matrix</b> $A$ , each $a_{ij}$ representing the probability of moving from state $i$ to state $j$ , s.t. $\sum_{j=1}^N a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \dots o_T$	a sequence of $T$ <b>observations</b> , each one drawn from a vocabulary $V = v_1, v_2, \dots, v_V$
$B = b_i(o_t)$	a sequence of <b>observation likelihoods</b> , also called <b>emission probabilities</b> , each expressing the probability of an observation $o_t$ being generated from a state $q_i$
$\pi = \pi_1, \pi_2, \dots, \pi_N$	an <b>initial probability distribution</b> over states. $\pi_i$ is the probability that the Markov chain will start in state $i$ . Some states $j$ may have $\pi_j = 0$ , meaning that they cannot be initial states. Also, $\sum_{i=1}^n \pi_i = 1$

# Gliederung

1 Was sind DoReCo und Urum?

- DoReCo
- Urum

2 Was ist ein Hidden Markov Model (HMM)?

3 HMM als POS-Tagger

4 Meine Implementierung eines HMM als POS-Tagger für Urum

# Beispiel eines HMM als POS-Tagger

- Beobachtungen = folgende Sätze:

# Beispiel eines HMM als POS-Tagger

- Beobachtungen = folgende Sätze:
  - "Der Kundin gefällt das Haus ."

# Beispiel eines HMM als POS-Tagger

- Beobachtungen = folgende Sätze:
  - "Der Kundin gefällt das Haus ."
  - "Das ist schön ."

# Beispiel eines HMM als POS-Tagger

- Beobachtungen = folgende Wort-Wortart-Paarungen:
  - "Das/D Haus/N gefällt/V der/D Kundin/N ./PNCT"
  - "Das/Pro ist/V schön/A ./PNCT"

# Beispiel eines HMM als POS-Tagger

- Beobachtungen = folgende Wort-Wortart-Paarungen:
  - "Das/D Haus/N gefällt/V der/D Kundin/N ./PNCT"
  - "Das/Pro ist/V schön/A ./PNCT"
- Zustände = {D,N,V,A,Pro,PNCT}

# Beispiel eines HMM als POS-Tagger

- Beobachtungen = folgende Wort-Wortart-Paarungen:
  - "Das/D Haus/N gefällt/V der/D Kundin/N ./PNCT"
  - "Das/Pro ist/V schön/A ./PNCT"
- Zustände = {D,N,V,A,Pro,PNCT}
- Emissionswahrscheinlichkeiten:

# Beispiel eines HMM als POS-Tagger

- Beobachtungen = folgende Wort-Wortart-Paarungen:
  - "Das/D Haus/N gefällt/V der/D Kundin/N ./PNCT"
  - "Das/Pro ist/V schön/A ./PNCT"
- Zustände = {D,N,V,A,Pro,PNCT}
- Emissionswahrscheinlichkeiten:  $P(W|T) = C(\langle W, T \rangle)/C(T)$

# Beispiel eines HMM als POS-Tagger

- Beobachtungen = folgende Wort-Wortart-Paarungen:
  - "Das/D Haus/N gefällt/V der/D Kundin/N ./PNCT"
  - "Das/Pro ist/V schön/A ./PNCT"
- Zustände = {D,N,V,A,Pro,PNCT}
- Emissionswahrscheinlichkeiten:  $P(W|T) = C(\langle W, T \rangle)/C(T)$

	D	N	V	A	Pro	PNCT
Das	<b>1/2</b>	0	0	0	<b>1/1</b>	0
Haus	0	<b>1/2</b>	0	0	0	0
gefällt	0	0	<b>1/2</b>	0	0	0
der	<b>1/2</b>	0	0	0	0	0
Kundin	0	<b>1/2</b>	0	0	0	0
ist	0	0	<b>1/2</b>	0	0	0
schön	0	0	0	<b>1/1</b>	0	0
.	0	0	0	0	0	<b>2/2</b>

# Beispiel eines HMM als POS-Tagger

- Transitionswahrscheinlichkeiten:

# Beispiel eines HMM als POS-Tagger

- Transitionswahrscheinlichkeiten:

$$P(T_2|T_1) = C(\langle T_1, T_2 \rangle) / C(T_1)$$

# Beispiel eines HMM als POS-Tagger

- Transitionswahrscheinlichkeiten:

$$P(T_2|T_1) = C(\langle T_1, T_2 \rangle) / C(T_1)$$

Das/D Haus/N gefällt/V der/D Kundin/N ./PNCT  
Das/Pro ist/V schön/A ./PNCT

# Beispiel eines HMM als POS-Tagger

- Transitionswahrscheinlichkeiten:

$$P(T_2|T_1) = C(\langle T_1, T_2 \rangle) / C(T_1)$$

- <S>Das/D Haus/N gefällt/V der/D Kundin/N ./PNCT<E>
- <S>Das/Pro ist/V schön/A ./PNCT<E>

# Beispiel eines HMM als POS-Tagger

- Transitionswahrscheinlichkeiten:

$$P(T_2|T_1) = C(\langle T_1, T_2 \rangle) / C(T_1)$$

- $\langle S \rangle$ Das/D Haus/N gefällt/V der/D Kundin/N ./PNCT $\langle E \rangle$
- $\langle S \rangle$ Das/Pro ist/V schön/A ./PNCT $\langle E \rangle$

	$\langle S \rangle$	D	N	V	A	Pro	PNCT	$\langle E \rangle$
$\langle S \rangle$	0	<b>1/2</b>	0	0	0	<b>1/2</b>	0	0
D	0	0	<b>2/2</b>	0	0	0	0	0
N	0	0	0	<b>1/2</b>	0	0	<b>1/2</b>	0
V	0	<b>1/2</b>	0	0	<b>1/2</b>	0	0	0
A	0	0	0	0	0	0	<b>1/1</b>	0
Pro	0	0	0	<b>1/1</b>	0	0	0	0
PNCT	0	0	0	0	0	0	0	<b>2/2</b>
$\langle E \rangle$	0	0	0	0	0	0	0	0

## Beispielberechnung

- <S> Das Haus ist schön . <E>.

# Beispielberechnung

- <S> Das Haus ist schön . <E>.
- <S>-D
  - 1/2\*1/2

# Beispielberechnung

- <S> Das Haus ist schön . <E>.
- <S>-D
  - $1/2 * 1/2$
- <S>-Pro
  - $1/2 * 1$

# Beispielberechnung

- <S> Das Haus ist schön . <E>.
- <S>-D-N
  - $1/2 * 1/2 * 1 * 1/2$
- <S>-Pro
  - $1/2 * 1$

# Beispielberechnung

- <S> Das Haus ist schön . <E>.
- <S>-D-N
  - $1/2 * 1/2 * 1 * 1/2$
- <S>-Pro-N
  - $1/2 * 1 * \mathbf{0} * 1/2 = 0$

# Beispielberechnung

- <S> Das Haus ist schön . <E>.
- <S>-D-N-V-A-.-<E>
  - $1/2 * 1/2 * 1 * 1/2 * 1/2 * 1/2 * 1 * 1 * 1 * 1 = 0,015625$
- <S>-Pro-N
  - $1/2 * 1 * \mathbf{0} * 1/2 = 0$

# Gliederung

- 1** Was sind DoReCo und Urum?
  - DoReCo
  - Urum
- 2** Was ist ein Hidden Markov Model (HMM)?
- 3** HMM als POS-Tagger
- 4** Meine Implementierung eines HMM als POS-Tagger für Urum

# 1. Skript: Wort-POS-Tag-Paare von Urum erhalten

Name	Größe	Detaillierter Dateityp	Änderungsdatum
doreco_CONVENTIONS.txt	3,0 kB	Einfaches Textdokument	2022-12-13
doreco_README.txt	2,7 kB	Einfaches Textdokument	2022-12-13
doreco_urum1249_dataset-info.txt	1,8 kB	Einfaches Textdokument	2022-12-13
doreco_urum1249_gloss-abbreviations.csv	958 Bytes	CSV-Dokument	2022-12-13
doreco_urum1249_metadata.csv	27,5 kB	CSV-Dokument	2022-12-13
doreco_urum1249_ph.csv	24,9 MB	CSV-Dokument	2022-12-13
doreco_urum1249_renamed-tiers.csv	80,2 kB	CSV-Dokument	2022-12-13
doreco_urum1249_transcription-contains.csv	412 Bytes	CSV-Dokument	2022-12-13
doreco_urum1249_UUM-TXT-AN-0001.xml	241,4 kB	ELAN Annotation File	2022-12-13
doreco_urum1249_UUM-TXT-AN-0002.xml	4,6 kB	XML-Dokument	2024-02-10
doreco_urum1249_UUM-TXT-AN-0003.xml	88,1 kB	Einfaches Textdokument	2022-12-13
doreco_urum1249_UUM-TXT-AN-0004.xml	131,9 kB	XML-Dokument	2022-12-13
doreco_urum1249_UUM-TXT-AN-0005.xml	227,6 kB	ELAN Annotation File	2022-12-13
doreco_urum1249_UUM-TXT-AN-0006.xml	83,2 kB	Einfaches Textdokument	2022-12-13
doreco_urum1249_UUM-TXT-AN-0007.xml	120,0 kB	XML-Dokument	2022-12-13
doreco_urum1249_UUM-TXT-AN-0008.xml	366,4 kB	ELAN Annotation File	2022-12-13

748 Objekte ausgewählt (139,6 MB), Freier Speicherplatz: 65,0 GB

# 1. Skript: Wort-POS-Tag-Paare von Urum erhalten

```
scripts > 🐍 get_urum_data_words.py > ...
+
5   from corflow import fromElan
6   import glob
7   import re
8
9
10
11
12
13
14
15
16
17 #ACTUAL-OPERATIONS-BEGIN-HERE#
18 input_files = "../input_files/urum_files/"
19 tagged_words_file_path = "../data/tagged_words_urum_words.txt"
20 eaf_files = glob.glob(input_files+"*.eaf")
21
22 tagged_words = []
23 #ELAN-files:
24 for file in eaf_files:
25     print(f"File name: {file.replace(input_files,'')}")
26     trans = fromElan.fromElan(file,encoding="utf-8")
27     #Getting the ref, and respective wd (and mb) and pos tier:
28     for ref_tier in find_tiers(trans,"ref@"):
29         for ch_tier in ref_tier.children():
30             if ch_tier.name.startswith("wd@"):
31                 wd_tier = ch_tier
32                 break
33             mb_tier = wd_tier.children()[0]
34             for ch_ch_tier in mb_tier.children():
35                 if ch_ch_tier.name.startswith("ps@"):
36                     pos_tier = ch_ch_tier
37                     break
```

# Ebenen und Segmente in ELAN

	0001_DoReCo_doreco_urum1249_UUM-TXT-AN-00000-A01
ref@A01 [22]	To my mind, they came two hundred years ago from Trabzon.
ft@A01 [22]	bänä ğala gälđilär iki yuz el iräli trapezondadan
tx@A01 [22]	bänä   ğala   gälđilär   <p:>
wd@A01 [90]	bän   -ä   ğal   -a   gäl   -di   -lär   <p:>
mb@A01 [143]	1.SG   -D stay   -GER   come   -PS   -PL   <p:>
gl@A01 [143]	PN   -c   V   -tam   V   -ta   -num   <p:>
ps@A01 [143]	b   {   n   {   G   a   5   a   }   l   {   I   d   5   {   r   <p:>
ph@A01 [342]	****   **   ****   ****   ****   ***   ****   <p:>
doreco-mb-al [143]	1.SG -DAT stay -GER come -PST -PL two hundred year before Trabzon-LOC-ABL
ge-a@unknown [22]	

# 1. Skript: Wort-POS-Tag-Paare von Urum erhalten

```
38     .... #Acceptable-doreco-labels:  
39     .... doreco_labels = ["<<fm", "<<ui", "<<pr"]  
40     .... #Getting word-pos_tag-pairs:  
41     .... for ref_seg in ref_tier:  
42     ....     if ref_seg.content == "<p:>":  
43     ....         continue  
44     ....     if wd_tier in ref_seg.childDict():  
45     ....         for wd_seg in ref_seg.childDict()[wd_tier]:  
46     ....             wdseg_content = wd_seg.content  
47     ....             if wdseg_content == "<p:>":  
48     ....                 continue  
49     ....             elif wdseg_content.startswith("<<"):  
50     ....                 for label in doreco_labels:  
51     ....                     if label in wdseg_content:  
52     ....                         wdseg_content = wdseg_content.replace(label,"").replace(">","")  
53     ....             if wdseg_content.startswith("<<"):  
54     ....                 continue  
55     ....             if mb_tier in wd_seg.childDict():  
56     ....                 for mb_seg in wd_seg.childDict()[mb_tier]:  
57     ....                     if pos_tier in mb_seg.childDict():  
58     ....                         pos_seg = mb_seg.childDict()[pos_tier][0]  
59     ....                         if pos_seg.content == "":  
60     ....                             continue
```

# 1. Skript: Wort-POS-Tag-Paare von Urum erhalten

```
61     if re.search("[A-WY-Z]",pos_seg.content.replace("-",""))[0]):  
62         wdseg_content = wdseg_content.strip()  
63         tagged_words.append((wdseg_content,pos_seg.content.replace("-","")))  
64         break  
65     if tagged_words[-1][-1] != "<E>":  
66         tagged_words.append("<E>,<E>")  
  
68 #Saving all pairs of word-(pos)tag with ';' as a delimiter in a separate csv file.  
69 with open(tagged_words_file_path,"w") as file:  
70     for w,t in tagged_words:  
71         file.write(w+";"+t+"\n")  
data >  tagged_words_urum_words.txt
```

```
1  bizim;PN  
2  halh;N  
3  čoğ;A  
4  işlier;V  
5  <E>;<E>  
6  gürjülär;N  
7  sävmerlär;V  
8  išlämäh;V  
9  onnar;PN  
10 emäh;N  
11 iślär;N  
12 säverlär;V  
13 <E>;<E>  
14 bizim;PN  
15 halh;N  
16 čoğ;A
```

```
21187 <E>;<E>  
21188 čaimız;N  
21189 var;V  
21190 yazın;N  
21191 gidirih;V  
21192 balih;N  
21193 dutmaya;V  
21194 atdıhat;N  
21195 edirih;V  
21196 <E>;<E>  
21197
```

## 2. Skript: HMM trainieren und testen

```
scripts > 🐍 hmm_v2.py > ...
223 #ACTUAL-OPERATIONS-BEGIN-HERE#
224 #tagged_words_file_path = "../data/tagged_words_urum_words.txt"
225 tagged_words_file_path = "../data/tagged_words_urum_words.txt"
226
227 #Get the word-tag-pairs from the respective file as tuples inside a list.
228 tagged_words_l = get_csv_as_list(tagged_words_file_path, ";")
229
230 #Put word-tag-pairs into lists representing sentences and put them in one list.
231 tagged_sentences = get_sent_for_wd_tag_pairs_urum(tagged_words_l)
232 #tagged_sentences = get_sent_for_wd_tag_pairs(tagged_words_l)

34 def get_csv_as_list(file_path, separator,):
35     '''Loads a csv file from *file_path* with *sep* as the delimiter and returns it as list
36     with tuples, in which every tuple represents a row and every value in a tuple represents a
37     value of a column.'''
38     tagged_words = pd.read_csv(file_path, sep=separator, header=None)
39     tagged_words = tagged_words.dropna()
40     tagged_words_l = tagged_words.values.tolist()
41     return [(i[0], i[1]) for i in tagged_words_l]
```

## 2. Skript: HMM trainieren und testen

```
55 def get_sent_for_wd_tag_pairs_urum(wd_tag_pairs:list):
56     """Returns a list of sentences, where a sentence is represented as a list containing
57     any number of word-tag-pairs (as tuples) from a given list of word-tag-pairs.
58     *wd_tag_pairs*(as tuples). If *wd_tag_pairs* contains list of word-tag-pairs rather
59     than tuples, the former get transformed to tuples."""
60     pairs = wd_tag_pairs
61     if any(isinstance(pair, list) for pair in wd_tag_pairs):
62         pairs = [(i[0], i[1]) for i in wd_tag_pairs]
63     tagged_sentences = []
64     single_sentence = []
65     for wd, tag in pairs:
66         if wd != "<E>":
67             single_sentence.append((wd, tag))
68         else:
69             tagged_sentences.append(single_sentence)
70             single_sentence = []
71     return tagged_sentences
```

```
[[('bizim', 'PN'), ('halh', 'N'), ('čoğ', 'A'), ('išlier', 'V')], [('gürjülär', 'N'), ('sävmerlär', 'V'),
    ('iślämäh', 'V'), ('onnar', 'PN'), ('emäh', 'N'), ('iślär', 'N'), ('säverlär', 'V')], [('bizim', 'PN'), ('halh',
    'N'), ('čoğ', 'A'), ('išlier', 'V'), ('yahši', 'A'), ('halhtır', 'N')], [('nä', 'PN'), ('diem', 'V'),
    ('čto', 'PN'), ('ešyo', 'A')], [('güzäl', 'A'), ('halh', 'N')], [('am', 'A'), ('bizim', 'PN'), ('halh',
    'N'), ('čauch', 'PN'), ('čačilješ', 'V'), ('büləşən', 'V'), ('küt', 'C'), ('bəyəndəst', 'M'), ('fədətələst',
```

## 2. Skript: HMM trainieren und testen

```
234 #Create the training and test data randomly (optionally with a seed) by default in the  
235 #ratio 8:2 (train:test) from the list of tagged sentences.  
236 train_sentences,test_sentences = split_train_test(tagged_sentences,seed=469283984701)  
237  
238 #Apply the hmm algorithm and print the accuracy of correctly predicted pos tags for the  
239 #test sentences in percent.  
240 hmm_accuracy = hmm_algorithm(train_sentences,test_sentences)  
241 print(f"The hmm's sentence prediction accuracy is {hmm_accuracy[0]*100}% and its tag  
242 prediction accuracy is {hmm_accuracy[1]*100}%.")
```

```
14 def split_train_test(data,size=0.8,seed=False):  
15     """Returns a tuple with two lists, the first being the training data and the second  
     being the test data based on randomly shuffling the *data* with a given random *seed*.  
     (by default no seed is given). The size of the training data equals *size* times its  
     length (by default 0.8*length of *data*). The rest of *data* amounts to the test data.  
     """  
16     if isinstance(seed,bool):  
17         rng = np.random.default_rng()  
18     elif isinstance(seed,int):  
19         rng = np.random.default_rng(seed)  
20     split_num = math.ceil(len(data)*size)  
21     random_data = data  
22     rng.shuffle(random_data)  
23     train = random_data[:split_num]  
24     test = random_data[split_num:]  
25     return (train,test)
```

## 2. Skript: HMM trainieren und testen

```
120 def hmm_algorithm(train_sentences,test_sentences):
121     '''Creates an hmm model of *train_sentences* and predicts *test_sentences* based on
122     that model. Returns a tuple with its first value being the accuracy of the hmm model
123     when predicting sentences and its second value being the accuracy of the hmm model when
124     predicting single tags (here: pos tags).'''
125     trans_prob_df = get_transition_probability(train_sentences)
126     test_sents = test_sentences
127     for sent in test_sents:
128         sent.insert(0, ("<S>","<S>"))
129         sent.append(("<E>","<E>"))
130     train_data = reduce_dim(train_sentences)
131     sentences_correctly_predicted = 0
132     tags_correctly_predicted = 0
133     for sent in test_sents:
134         sent_unpredictable = False
135         paths = []
136         if ind == (len(sent[1:])-1):
137             for tags_l,p in paths:
```

## 2. Skript: HMM trainieren und testen

```
100 def get_transition_probability(training_sentences):
101     """Return a pandas DataFrame representing the transition probability of all unique
102     (pos) tags (as well as the added start- and end-sentence tags '<S>' and '<E>') of a
103     given list *training_sentences*, whose lists contain single sentences of the training
104     data. The rows represent the first (left) tag (of a bigram), and the columns the second
105     (right) tag."""
106     new_sentences = training_sentences
107     for sent in new_sentences:
108         sent.insert(0, ("<S>","<S>"))
109         sent.append(("<E>","<E>"))
110     new_data = reduce_dim(new_sentences)
111     unique_tags = list(set([tag for wd, tag in new_data]))
112     trans_prob_arr = np.zeros((len(unique_tags), len(unique_tags)))
113     for ind1,t1 in enumerate(unique_tags):
114         for ind2,t2 in enumerate(unique_tags):
115             if t1 == "<E>":
116                 trans_prob_arr[ind1,ind2] = 0
117             elif t2 == "<S>":
118                 trans_prob_arr[ind1,ind2] = 0
119             else:
120                 tag1_tag2, tag1_x = get_transition_probability_base(t1,t2,new_data)
121                 trans_prob_arr[ind1,ind2] = tag1_tag2/tag1_x
122
123     return pd.DataFrame(trans_prob_arr,columns=unique_tags,index=unique_tags)
```

## 2. Skript: HMM trainieren und testen

```
88 def get_transition_probability_base(tag1,tag2,data):
89     '''Returns for calculating the transition probability of a pair of tags <tag1*,*tag2*>
90     (*tag1* occurs right before *tag2*) from a list of word-tag-pairs (tuples) *data* a
91     tuple with two ints.'''
92     #P(tag2|tag1) = C(<tag1,tag2>)/C(<tag1,X>)
93     tags = [tag for word,tag in data]
94     count_tag1_tag2 = 0
95     for ind in range(len(tags)-1):
96         if (tags[ind] == tag1) & (tags[ind+1] == tag2):
97             count_tag1_tag2 += 1
98     #count_tag1_tag2 = len([tag for ind,tag in enumerate(tags) if (tag == tag1) & (tags[ind+1] == tag2)])
99     count_tag1_x = len([tag for tag in tags if tag == tag1])
100    return (count_tag1_tag2, count_tag1_x)
```

## 2. Skript: HMM trainieren und testen

	N	<E>	PN	INT	AN	...	N-num	Pn.A	<S>	C/A	PN-case
N	0.215907	0.131880	0.088519	0.000225	0.0	...	0.000000	0.000899	0.0	0.000674	0.00000
<E>	0.000000	0.000000	0.000000	0.000000	0.0	...	0.000000	0.000000	0.0	0.000000	0.00000
PN	0.379246	0.033039	0.098651	0.000000	0.0	...	0.000000	0.001396	0.0	0.000465	0.00000
INT	0.333333	0.000000	0.000000	0.000000	0.0	...	0.000000	0.000000	0.0	0.000000	0.00000
AN	0.000000	0.000000	0.000000	0.000000	0.0	...	0.000000	0.000000	0.0	0.500000	0.00000
P	0.290000	0.050000	0.330000	0.000000	0.0	...	0.000000	0.000000	0.0	0.000000	0.00000
PRT	0.190476	0.088435	0.227891	0.000000	0.0	...	0.000000	0.000000	0.0	0.023810	0.00000
VV	0.000000	0.000000	0.000000	0.000000	0.0	...	0.000000	0.000000	0.0	0.000000	0.00000
INF	0.333333	0.000000	0.000000	0.000000	0.0	...	0.000000	0.000000	0.0	0.000000	0.00000
Pn.	0.000000	0.000000	0.000000	0.000000	1.0	...	0.000000	0.000000	0.0	0.000000	0.00000
PN?	0.000000	1.000000	0.000000	0.000000	0.0	...	0.000000	0.000000	0.0	0.000000	0.00000
Q	0.482529	0.044925	0.046589	0.000000	0.0	...	0.000000	0.000000	0.0	0.001664	0.00000
REF.PN	0.000000	0.000000	0.000000	0.000000	0.0	...	0.000000	0.000000	0.0	0.000000	0.00000
V	0.228171	0.202652	0.135101	0.000250	0.0	...	0.000000	0.000250	0.0	0.000500	0.00025
C	0.206154	0.021538	0.226154	0.000000	0.0	...	0.000000	0.001538	0.0	0.004615	0.00000
A/C/PN	0.290323	0.032258	0.072581	0.000000	0.0	...	0.000000	0.000000	0.0	0.000000	0.00000
A	0.295626	0.085219	0.085973	0.000000	0.0	...	0.000377	0.000000	0.0	0.000754	0.00000
PR	0.347826	0.043478	0.000000	0.000000	0.0	...	0.000000	0.000000	0.0	0.000000	0.00000
N-num	0.000000	0.000000	0.000000	0.000000	0.0	...	0.000000	0.000000	0.0	0.000000	0.00000
Pn.A	0.555556	0.000000	0.000000	0.000000	0.0	...	0.000000	0.000000	0.0	0.000000	0.00000
<S>	0.230337	0.000562	0.272472	0.000562	0.0	...	0.000000	0.000000	0.0	0.007303	0.00000
C/A	0.272727	0.212121	0.181818	0.000000	0.0	...	0.000000	0.000000	0.0	0.000000	0.00000
PN-case	0.000000	0.000000	0.000000	0.000000	0.0	...	0.000000	0.000000	0.0	0.000000	0.00000

[23 rows x 23 columns]

## 2. Skript: HMM trainieren und testen

```
137     for tags_l,p in paths:
138         trans_p = trans_prob_df.at[tags_l[-1], "<E>"]
139         if trans_p == 0:
140             continue
141         current_tags = tags_l + ["<E>"]
142         current_p = p * trans_p
143         current_paths.append((current_tags, current_p))
144     elif (ind > 0) and (paths):
145         unique_tags, num_unique_tags = get_tag_types(word, train_data)
146         if num_unique_tags == 0:
147             continue
148         for tag in unique_tags:
149             wd_tag_c, tag_c = get_emission_probability_base(word, tag, train_data)
150             emis_p = wd_tag_c / tag_c
151             for tags_l,p in paths:
152                 trans_p = trans_prob_df.at[tags_l[-1], tag]
153                 if trans_p == 0:
154                     continue
155                 current_tags = tags_l + [tag]
156                 current_p = p * emis_p * trans_p
157                 current_paths.append((current_tags, current_p))
158     elif ind == 0:
159         unique_tags, num_unique_tags = get_tag_types(word, train_data)
160         if num_unique_tags == 0:
161             continue
162         for tag in unique_tags:
163             wd_tag_c, tag_c = get_emission_probability_base(word, tag, train_data)
```

## 2. Skript: HMM trainieren und testen

```
163     wd_tag_c,tag_c = get_emission_probability_base(word,tag,train_data)
164     emis_p = wd_tag_c/tag_c
165     trans_p = trans_prob_df.at["<S>",tag]
166     if trans_p == 0:
167         continue
168     current_tags = [<S>,tag]
169     current_p = trans_p * emis_p
170     current_paths.append((current_tags,current_p))
171     #If during the calculation of all paths p.value, no path has a trans_p > 0,
172     #therefore, no path has any predictive value, the sentence is not predictable
173     #and therefore skipped.
174     elif not paths:
175         sent_unpredictable = True
176         break
177     paths = current_paths
178     #Check, whether a word was in the test but not in the train sentences. If so, skip
179     #this sentence.
180     if (sent_unpredictable) | (not paths):
181         continue
182     #print(f"computed paths: {paths}")
183     optimal_path = max(paths,key=itemgetter(1))
184     #print(f"optimal path: {optimal_path}")
185     actual_path = [tag for wd,tag in sent]
```

## 2. Skript: HMM trainieren und testen

```
71 def get_emission_probability_base(word:str,tag:str,data):
72     '''Returns a tuple containing the emission probability of a word given a tag from a
73     list of word-tag-pairs (tuples) *data*. A tuple with two ints.'''
74     #P(word|tag) := (P(word) · n · P(tag)) / P(tag)
75     #P(word|tag) := C(<tag,word>) / C(<tag,X>)
76     pairs_with_tag = [(wd,t) for wd,t in data if t == tag]
77     tag_count = len(pairs_with_tag)
78     word_tag_count = len([wd for wd,t in pairs_with_tag if wd == word])
79     return (word_tag_count,tag_count)

80 def get_tag_types(word:str,data):
81     '''Returns a tuple containing i) a list with unique tags for a given word of a list
82     of tuples of word-tag-pairs *data* and ii) its length.'''
83     unique_tags = set()
84     for wd,tag in data:
85         if wd == word:
86             unique_tags.add(tag)
87     return (list(unique_tags),len(unique_tags))
```

## 2. Skript: HMM trainieren und testen

```
185     actual_path = [tag for wd, tag in sent]
186     #Check, whether the predictions are correct or not.
187     #print(f"actual path: {actual_path}")
188     if optimal_path[0] == actual_path:
189         sentences_correctly_predicted += 1
190         tags_correctly_predicted += len(actual_path) - 2
191     else:
192         for index in range(1, len(optimal_path[0]) - 1):
193             if optimal_path[0][index] == actual_path[index]:
194                 tags_correctly_predicted += 1
195
196     sentence_prediction_accuracy = sentences_correctly_predicted / len(test_sentences)
197     tag_prediction_accuracy = tags_correctly_predicted / len(reduce_dim(test_sentences))
198
199     return sentence_prediction_accuracy, tag_prediction_accuracy
```

The hmm's sentence prediction accuracy is 20.945945945945947% and its tag prediction accuracy is 36.82219419924338%.

```
real    0m20.703s
user    0m20.676s
sys     0m0.228s
```

# Referenzen

- Jurafsky, Daniel and Martin, James H. (2024): Speech and Language Processing. Draft of February 3, 2024.
- Great Learning Team (2022): Part of Speech (POS) tagging with Hidden Markov Model. URL:  
 [<https://www.mygreatlearning.com/blog/pos-tagging/>](https://www.mygreatlearning.com/blog/pos-tagging/)  
(zuletzt aufgerufen am 10.02.2024).
- Skopeteas, Stavros (2015): Caucasian Urums and Urum language.
- DoReCo Projekt Webseite. URL:  [<https://doreco.info/>](https://doreco.info/)  
(zuletzt aufgerufen am 10.02.2024).
- DoReCo Sprachdaten Webseite. URL:  
 [<https://doreco.huma-num.fr>](https://doreco.huma-num.fr) (zuletzt aufgerufen am 10.02.2024).