NLP for small, endangered languages: Building an HMM (Hidden Markov Model) as a POS-Tagger (for Urum)

Aleksandr Schamberger

Humboldt University Berlin
Department of German Studies and Linguistics
Colloquium Syntax and Semantic
Summer Semester 2024

May 26, 2024

Outline

Outline

Motivation and Purpose

- Motivation:

Motivation and Purpose

- Motivation:
  - Use NLP (Natural Language Processing) techniques to work with linguistic data.

Motivation and Purpose

- Motivation:
  - Use NLP (Natural Language Processing) techniques to work with linguistic data.
  - Try to build something yourself and have an in-depth view into a model.

Motivation and Purpose

- Motivation:
  - Use NLP (Natural Language Processing) techniques to work with linguistic data.
  - Try to build something yourself and have an in-depth view into a model.
  - Use those techniques to work with and explore linguistic data from small, endangered languages.

Motivation and Purpose

- Motivation:
  - Use NLP (Natural Language Processing) techniques to work with linguistic data.
  - Try to build something yourself and have an in-depth view into a model.
  - Use those techniques to work with and explore linguistic data from small, endangered languages.
- Purpose (Goals):

Motivation and Purpose

- Motivation:
  - Use NLP (Natural Language Processing) techniques to work with linguistic data.
  - Try to build something yourself and have an in-depth view into a model.
  - Use those techniques to work with and explore linguistic data from small, endangered languages.
- Purpose (Goals):
  - Build an easy, purely stochastic model and train it on linguistic data from one such small, endangered language (Urum).

Motivation and Purpose

- Motivation:
  - Use NLP (Natural Language Processing) techniques to work with linguistic data.
  - Try to build something yourself and have an in-depth view into a model.
  - Use those techniques to work with and explore linguistic data from small, endangered languages.
- Purpose (Goals):
  - Build an easy, purely stochastic model and train it on linguistic data from one such small, endangered language (Urum).
  - Use this model to predict the part-of-speech of words of this small language. Do the same for words of a common language (English).

Motivation and Purpose

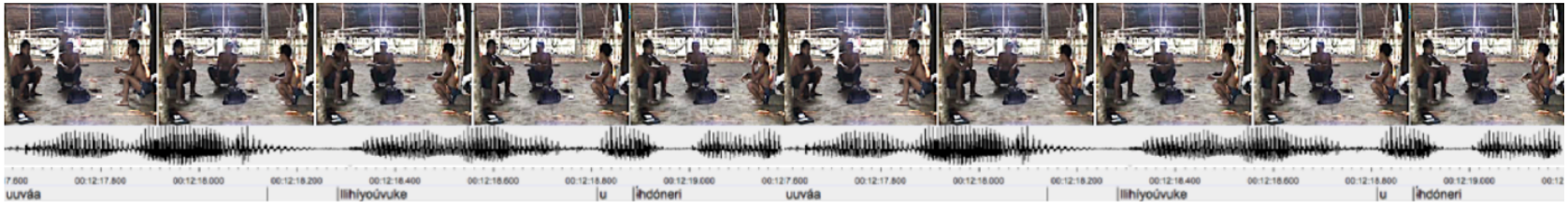
- Motivation:
  - Use NLP (Natural Language Processing) techniques to work with linguistic data.
  - Try to build something yourself and have an in-depth view into a model.
  - Use those techniques to work with and explore linguistic data from small, endangered languages.
- Purpose (Goals):
  - Build an easy, purely stochastic model and train it on linguistic data from one such small, endangered language (Urum).
  - Use this model to predict the part-of-speech of words of this small language. Do the same for words of a common language (English).
  - Compare the results (prediction accuracy of the model) from both languages and try to interpret the findings.

Outline

DoReCo: Language Documentation Reference Corpus

DoReCo

Language Documentation Reference Corpus



- Website of the project: https://doreco.info/
- Website of the repository: 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
- "time-aligned multilingual reference corpus built from documentations of [51 small, endangered] languages" (Frank Seifert)

DoReCo: Language Documentation Reference Corpus

- DoReCo: Language **D**ocumentation **R**eference **C**orpus
- "time-aligned multilingual reference corpus built from documentations of [51 small, endangered] languages" (Frank Seifert)
- spoken language data with audio recordings of mostly narrative texts

DoReCo: Language Documentation Reference Corpus

- DoReCo: Language **D**ocumentation **R**eference **C**orpus
- "time-aligned multilingual reference corpus built from documentations of [51 small, endangered] languages" (Frank Seifert)
- spoken language data with audio recordings of mostly narrative texts
- phonetic transcriptions are time-aligned for all 51 languages

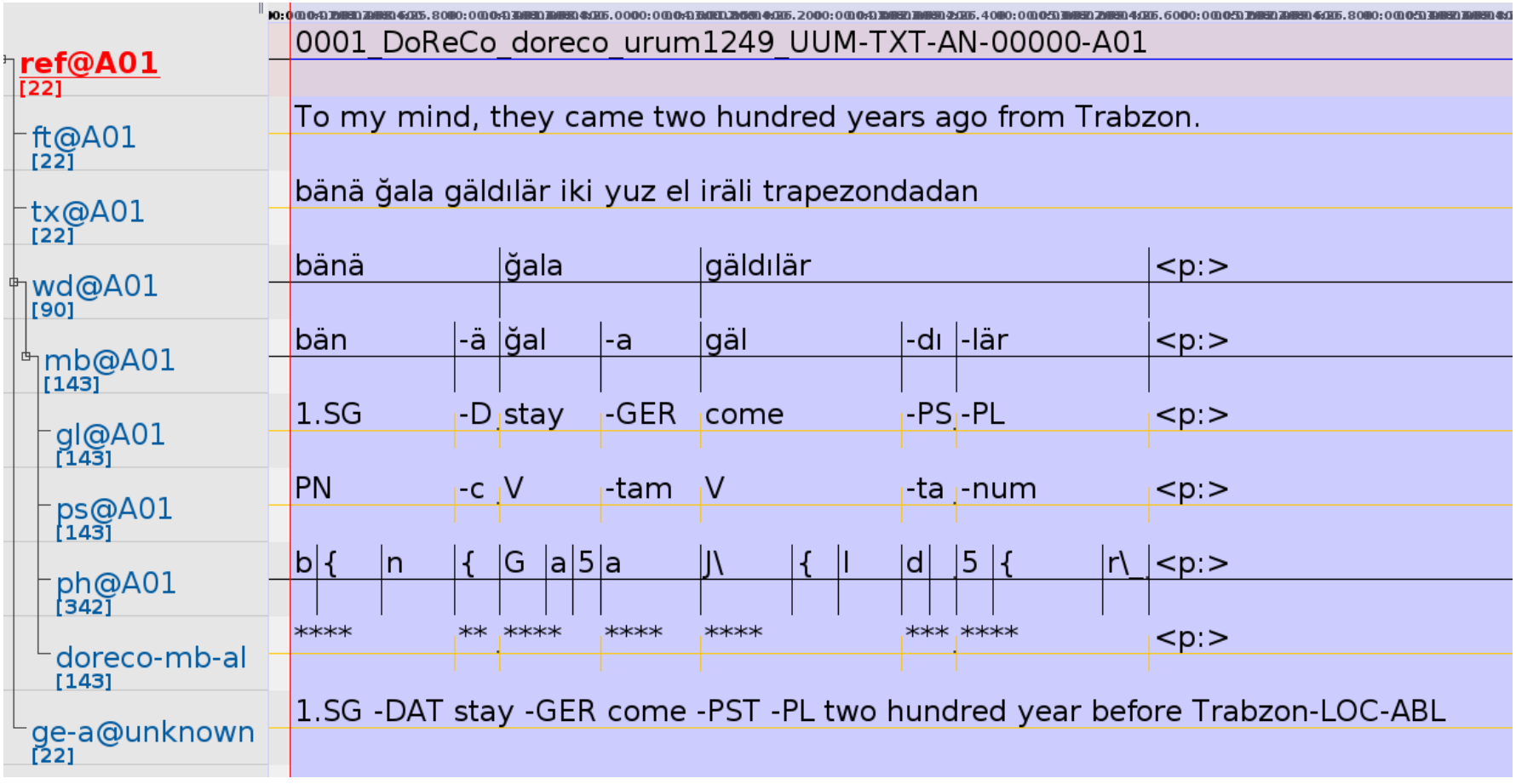
DoReCo: Language Documentation Reference Corpus

- DoReCo: Language **D**ocumentation **R**eference **C**orpus
- "time-aligned multilingual reference corpus built from documentations of [51 small, endangered] languages" (Frank Seifert)
- spoken language data with audio recordings of mostly narrative texts
- phonetic transcriptions are time-aligned for all 51 languages
- time-aligned morphological transcriptions for 38 languages

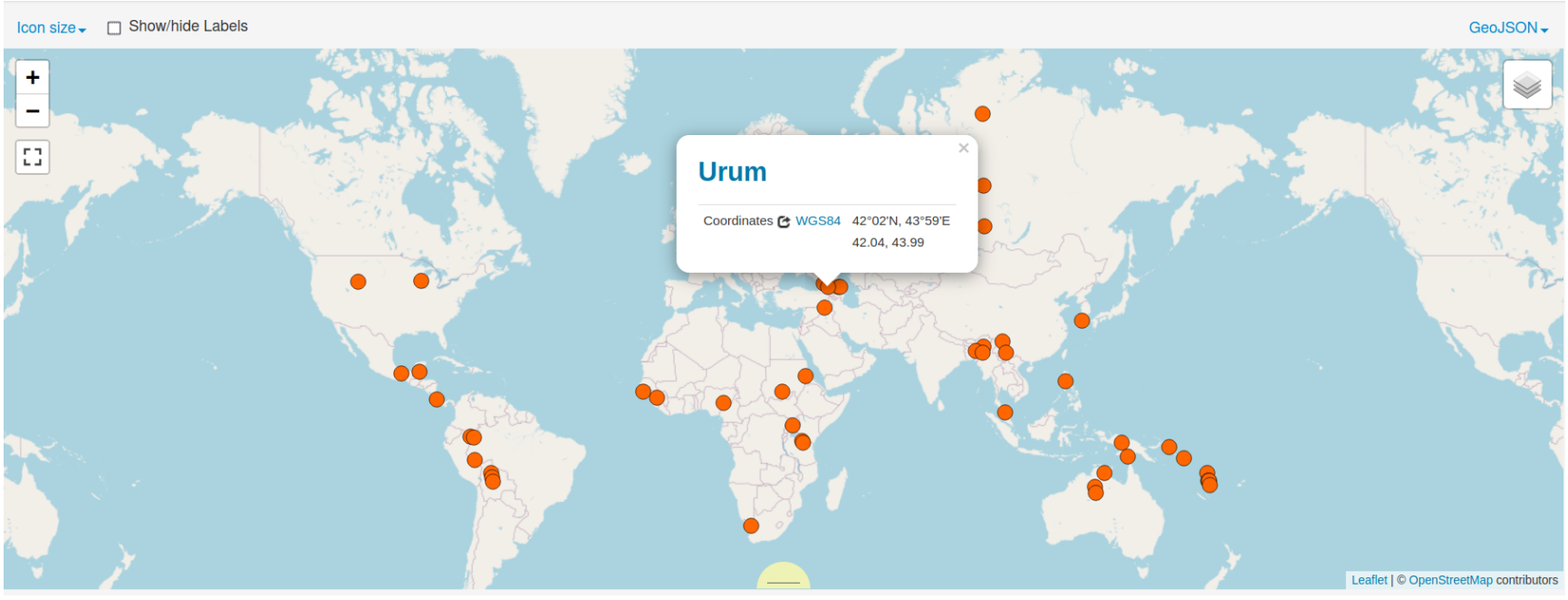
DoReCo: Language Documentation Reference Corpus

- DoReCo: Language **D**ocumentation **R**eference **C**orpus
- "time-aligned multilingual reference corpus built from documentations of [51 small, endangered] languages" (Frank Seifert)
- spoken language data with audio recordings of mostly narrative texts
- phonetic transcriptions are time-aligned for all 51 languages
- time-aligned morphological transcriptions for 38 languages
- data (annotations) is available as eaf (ELAN), TextGrid and csv files

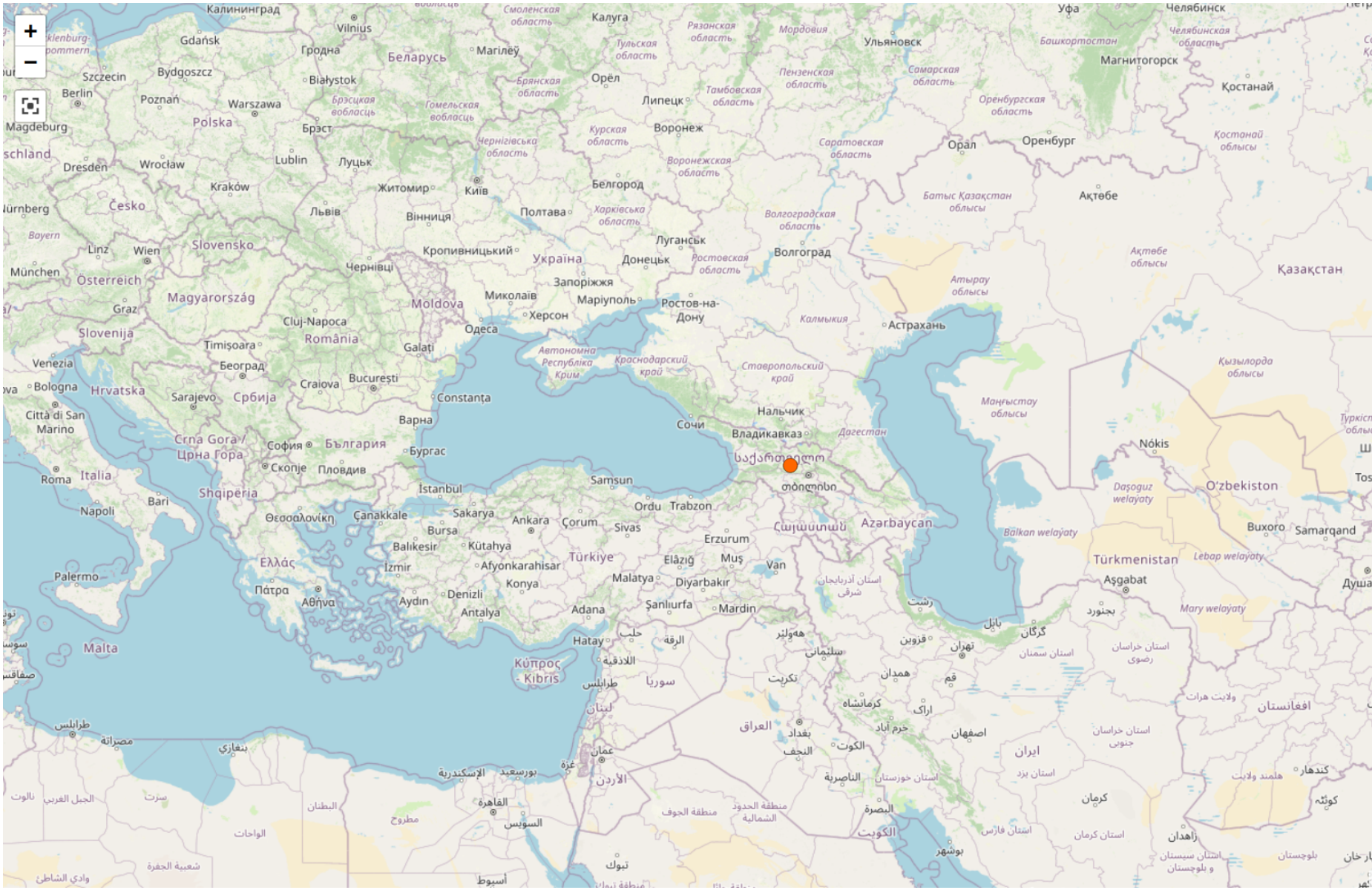
DoReCo: Example in ELAN



Urum: DoReCo Map



Urum: Map: More Detailed



Urum

- Caucasian Urum (not Crimean Urum)

Urum

- Caucasian Urum (not Crimean Urum)
- Turkish language with Russian loanwords spoken in Georgia

Urum

- Caucasian Urum (not Crimean Urum)
- Turkish language with Russian loanwords spoken in Georgia
- "Turkish-speaking Greek populations of Northeastern Anatolia" (Stavros Skopeteas: 2015) immigrating to the Caucasus in the beginning of the 19th century

Urum

- Caucasian Urum (not Crimean Urum)
- Turkish language with Russian loanwords spoken in Georgia
- "Turkish-speaking Greek populations of Northeastern Anatolia" (Stavros Skopeteas: 2015) immigrating to the Caucasus in the beginning of the 19th century
- People with greek ethnicity around the lake Tsalka (most of them speak Urum):

Urum

- Caucasian Urum (not Crimean Urum)
- Turkish language with Russian loanwords spoken in Georgia
- "Turkish-speaking Greek populations of Northeastern Anatolia" (Stavros Skopeteas: 2015) immigrating to the Caucasus in the beginning of the 19th century
- People with greek ethnicity around the lake Tsalka (most of them speak Urum):
  - 1979: 30.811

Urum

- Caucasian Urum (not Crimean Urum)
- Turkish language with Russian loanwords spoken in Georgia
- "Turkish-speaking Greek populations of Northeastern Anatolia" (Stavros Skopeteas: 2015) immigrating to the Caucasus in the beginning of the 19th century
- People with greek ethnicity around the lake Tsalka (most of them speak Urum):
  - 1979: 30.811
  - 2002: 4.589

Urum

- Caucasian Urum (not Crimean Urum)
- Turkish language with Russian loanwords spoken in Georgia
- "Turkish-speaking Greek populations of Northeastern Anatolia" (Stavros Skopeteas: 2015) immigrating to the Caucasus in the beginning of the 19th century
- People with greek ethnicity around the lake Tsalka (most of them speak Urum):
  - 1979: 30.811
  - 2002: 4.589
  - 2005: 1.500

Urum

- Caucasian Urum (not Crimean Urum)
- Turkish language with Russian loanwords spoken in Georgia
- "Turkish-speaking Greek populations of Northeastern Anatolia" (Stavros Skopeteas: 2015) immigrating to the Caucasus in the beginning of the 19th century
- People with greek ethnicity around the lake Tsalka (most of them speak Urum):
  - 1979: 30.811
  - 2002: 4.589
  - 2005: 1.500
- "data [...] were collected with native speakers in Tsalka and Tbilisi between 2009 and 2013"

Why Urum?

- small, endangered language

Why Urum?

- small, endangered language
- can be compared with already existing models for the Turkish language

Why Urum?

- small, endangered language
- can be compared with already existing models for the Turkish language
- the data has pos-tags for words but also for morphs

Why Urum?

- small, endangered language
- can be compared with already existing models for the Turkish language
- the data has pos-tags for words but also for morphs
- the data looks clean

Brown Corpus

- "[...] the first million-word electronic corpus of English, created in 1961 at Brown University. This corpus contains text from 500 sources, and the sources have been categorized by genre [...]" (NLTK book chapter 2; Steven Bird: 2019)

Brown Corpus

- "[...] the first million-word electronic corpus of English, created in 1961 at Brown University. This corpus contains text from 500 sources, and the sources have been categorized by genre [...]" (NLTK book chapter 2; Steven Bird: 2019)
- news genre: Chicago Tribune: Society Reportage

Outline

What is a Hidden Markov Model (HMM)?

- H = Hidden
- M = Markov
- M = Model

What is a Hidden Markov Model (HMM)?

- H = Hidden
- M = Markov
- M = Model
  - stochastic model predicting a sequence of (future) states (so called **Markov chains**). E.g. the weather, if a bus arrives at a certain time.

What is a Hidden Markov Model (HMM)?

- H = Hidden
- M = Markov
  - Andrei Andrejewitsch Markow, russian mathematician (1856-1922); prediction of sequences of states is based on the following property:
- M = Model
  - stochastic model predicting a sequence of (future) states (so called **Markov chains**). E.g. the weather, if a bus arrives at a certain time.

What is a Hidden Markov Model (HMM)?

- H = Hidden
- M = Markov
  - Andrei Andrejewitsch Markow, russian mathematician (1856-1922); prediction of sequences of states is based on the following property:
  - **Markov property**: $P(q_i = a|q_1...q_{i-1}) = P(q_i = a|q_{i-1})$
- M = Model
  - stochastic model predicting a sequence of (future) states (so called **Markov chains**). E.g. the weather, if a bus arrives at a certain time.

What is a Hidden Markov Model (HMM)?

- H = Hidden

- M = Markov
  - Andrei Andrejewitsch Markow, russian mathematician (1856-1922); prediction of sequences of states is based on the following property:
  - **Markov property**: $P(q_i = a|q_1...q_{i-1}) = P(q_i = a|q_{i-1})$ the probability of a state q_i occuring inside a sequence of states $q_1...q_{i-1}$ depends only on the probability of the latest occuring state q_{i-1}
- M = Model
  - stochastic model predicting a sequence of (future) states (so called **Markov chains**). E.g. the weather, if a bus arrives at a certain time.

What is a Hidden Markov Model (HMM)?

- H = Hidden
  - the states themselves are not directly observable, but hidden.

- M = Markov
  - Andrei Andrejewitsch Markow, russian mathematician (1856-1922); prediction of sequences of states is based on the following property:
  - **Markov property**: $P(q_i = a|q_1...q_{i-1}) = P(q_i = a|q_{i-1})$ the probability of a state q_i occuring inside a sequence of states $q_1...q_{i-1}$ depends only on the probability of the latest occuring state q_{i-1}
- M = Model
  - stochastic model predicting a sequence of (future) states (so called **Markov chains**). E.g. the weather, if a bus arrives at a certain time.

What is a Hidden Markov Model (HMM)?

- H = Hidden
  - the states themselves are not directly observable, but hidden.
  - the states are observed through other observations associated with them.
- M = Markov
  - Andrei Andrejewitsch Markow, russian mathematician (1856-1922); prediction of sequences of states is based on the following property:
  - **Markov property**: $P(q_i = a|q_1...q_{i-1}) = P(q_i = a|q_{i-1})$ the probability of a state q_i occuring inside a sequence of states $q_1...q_{i-1}$ depends only on the probability of the latest occuring state q_{i-1}
- M = Model
  - stochastic model predicting a sequence of (future) states (so called **Markov chains**). E.g. the weather, if a bus arrives at a certain time.

Hidden Markov Model

$Q = q_1 q_2 \ldots q_N$	a set of N **states**
$A = a_{11} \ldots a_{ij} \ldots a_{NN}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \ldots o_T$	a sequence of T **observations**, each one drawn from a vocabulary $V = v_1, v_2, ..., v_V$
$B = b_i(o_t)$	a sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state q_i
$\pi = \pi_1, \pi_2, ..., \pi_N$	an **initial probability distribution** over states. π_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, $\sum_{i=1}^{n} \pi_i = 1$

Outline

Toy Example for an HMM as a POS-Tagger

- O: Observations:

Toy Example for an HMM as a POS-Tagger

- O: Observations:
  - *This house is beautiful.*
  - *I like your hair.*
  - *This is red.*

Toy Example for an HMM as a POS-Tagger

- O: Observations:
  - *This house is beautiful.*
  - *I like your hair.*
  - *This is red.*
- Q: States:

Toy Example for an HMM as a POS-Tagger

- O: Observations:
  - *This house is beautiful.*
  - *I like your hair.*
  - *This is red.*
- Q: States: {D,N,V,A,PRO,PNCT}

Toy Example for an HMM as a POS-Tagger

- O: Observations:
  - *This house is beautiful.*
  - *I like your hair.*
  - *This is red.*
- Q: States: {D,N,V,A,PRO,PNCT}
  - This/D house/N is/V beautiful/A ./PNCT
  - I/PRO like/V your/D hair/N ./PNCT
  - This/PRO is/V red/A ./PNCT

Toy Example for an HMM as a POS-Tagger

- O: Observations:
  - *This house is beautiful.*
  - *I like your hair.*
  - *This is red.*
- Q: States: {D,N,V,A,PRO,PNCT}
  - This/D house/N is/V beautiful/A ./PNCT
  - I/PRO like/V your/D hair/N ./PNCT
  - This/PRO is/V red/A ./PNCT
- B: Emission Probability

Toy Example for an HMM as a POS-Tagger

- O: Observations:
  - *This house is beautiful.*
  - *I like your hair.*
  - *This is red.*
- Q: States: {D,N,V,A,PRO,PNCT}
  - This/D house/N is/V beautiful/A ./PNCT
  - I/PRO like/V your/D hair/N ./PNCT
  - This/PRO is/V red/A ./PNCT
- B: Emission Probability
- A: Transition Probability

Toy Example for an HMM as a POS-Tagger

- O: Observations:
  - *This house is beautiful.*
  - *I like your hair.*
  - *This is red.*
- Q: States: {D,N,V,A,PRO,PNCT}
  - This/D house/N is/V beautiful/A ./PNCT
  - I/PRO like/V your/D hair/N ./PNCT
  - This/PRO is/V red/A ./PNCT
- B: Emission Probability
- A: Transition Probability
- π: Initial Probability Distribution

Emission Probability

- Sentence to predict: *This is beautiful*.

Emission Probability

- Sentence to predict: *This is beautiful*.
- Word-Tag-Pairs:

Emission Probability

- Sentence to predict: *This is beautiful*.
- Word-Tag-Pairs:
  - This/D house/N is/V beautiful/A ./PNCT
  - I/PRO like/V your/D hair/N ./PNCT
  - This/PRO is/V red/A ./PNCT

Emission Probability

- Sentence to predict: *This is beautiful*.
- Word-Tag-Pairs:
  - This/D house/N is/V beautiful/A ./PNCT
  - I/PRO like/V your/D hair/N ./PNCT
  - This/PRO is/V red/A ./PNCT
- B: Emission Probability:

Emission Probability

- Sentence to predict: *This is beautiful*.
- Word-Tag-Pairs:
  - This/D house/N is/V beautiful/A ./PNCT
  - I/PRO like/V your/D hair/N ./PNCT
  - This/PRO is/V red/A ./PNCT
- B: Emission Probability: $P(W|T) = C(\langle W, T \rangle)/C(T)$

Emission Probability

- Sentence to predict: *This is beautiful*.
- Word-Tag-Pairs:
  - This/D house/N is/V beautiful/A ./PNCT
  - I/PRO like/V your/D hair/N ./PNCT
  - This/PRO is/V red/A ./PNCT
- B: Emission Probability: $P(W|T) = C(\langle W, T \rangle)/C(T)$
- B(*This*):

Emission Probability

- Sentence to predict: *This is beautiful*.
- Word-Tag-Pairs:
  - This/D house/N is/V beautiful/A ./PNCT
  - I/PRO like/V your/D hair/N ./PNCT
  - This/PRO is/V red/A ./PNCT
- B: Emission Probability: $P(W|T) = C(\langle W, T \rangle)/C(T)$
- B(*This*):
  - $P(This|D) = 1/2$

Emission Probability

- Sentence to predict: *This is beautiful*.
- Word-Tag-Pairs:
  - This/D house/N is/V beautiful/A ./PNCT
  - I/PRO like/V your/D hair/N ./PNCT
  - This/PRO is/V red/A ./PNCT
- B: Emission Probability: $P(W|T) = C(\langle W, T \rangle)/C(T)$
- B(*This*):
  - $P(This|D) = 1/2$
  - $P(This|PRO) = 1/2$

Emission Probability

- Sentence to predict: *This is beautiful*.
- Word-Tag-Pairs:
  - This/D house/N is/V beautiful/A ./PNCT
  - I/PRO like/V your/D hair/N ./PNCT
  - This/PRO is/V red/A ./PNCT
- B: Emission Probability: $P(W|T) = C(\langle W, T \rangle)/C(T)$
- B(*This*):
  - $P(This|D) = 1/2$
  - $P(This|PRO) = 1/2$
  - $P(This|N) = 0$
  - $P(This|V) = 0$
  - $P(This|A) = 0$
  - $P(This|PNCT) = 0$

Transition Probability

- Sentence to predict: *This is beautiful*.
- Word-Tag-Pairs:
  - This/D house/N is/V beautiful/A ./PNCT
  - I/PRO like/V your/D hair/N ./PNCT
  - This/PRO is/V red/A ./PNCT

Transition Probability

- Sentence to predict: *This is beautiful*.
- Word-Tag-Pairs:
  - This/D house/N is/V beautiful/A ./PNCT
  - I/PRO like/V your/D hair/N ./PNCT
  - This/PRO is/V red/A ./PNCT
- A: Transition Probability:

Transition Probability

- Sentence to predict: *This is beautiful*.
- Word-Tag-Pairs:
  - This/D house/N is/V beautiful/A ./PNCT
  - I/PRO like/V your/D hair/N ./PNCT
  - This/PRO is/V red/A ./PNCT
- A: Transition Probability: $P(T_2|T_1) = C(\langle T_1, T_2 \rangle)/C(T_1)$

Transition Probability

- Sentence to predict: *This is beautiful*.
- Word-Tag-Pairs:
  - <S> This/D house/N is/V beautiful/A ./PNCT <E>
  - <S> I/PRO like/V your/D hair/N ./PNCT <E>
  - <S> This/PRO is/V red/A ./PNCT <E>
- A: Transition Probability: $P(T_2|T_1) = C(\langle T_1, T_2 \rangle)/C(T_1)$

Transition Probability

- Sentence to predict: *This is beautiful*.
- Word-Tag-Pairs:
  - <S> This/D house/N is/V beautiful/A ./PNCT <E>
  - <S> I/PRO like/V your/D hair/N ./PNCT <E>
  - <S> This/PRO is/V red/A ./PNCT <E>
- A: Transition Probability: $P(T_2|T_1) = C(\langle T_1, T_2 \rangle)/C(T_1)$
- A(<S>,D):

Transition Probability

- Sentence to predict: *This is beautiful*.
- Word-Tag-Pairs:
  - <S> This/D house/N is/V beautiful/A ./PNCT <E>
  - <S> I/PRO like/V your/D hair/N ./PNCT <E>
  - <S> This/PRO is/V red/A ./PNCT <E>
- A: Transition Probability: $P(T_2|T_1) = C(\langle T_1, T_2 \rangle)/C(T_1)$
- A(<S>,D):

- A(<S>,PRO):

Transition Probability

- Sentence to predict: *This is beautiful*.
- Word-Tag-Pairs:
  - <S> This/D house/N is/V beautiful/A ./PNCT <E>
  - <S> I/PRO like/V your/D hair/N ./PNCT <E>
  - <S> This/PRO is/V red/A ./PNCT <E>
- A: Transition Probability: $P(T_2|T_1) = C(\langle T_1, T_2 \rangle)/C(T_1)$
- A(<S>,D):
  - $P(D| < S >) = 1/3$
- A(<S>,PRO):

Transition Probability

- Sentence to predict: *This is beautiful*.
- Word-Tag-Pairs:
  - <S> This/D house/N is/V beautiful/A ./PNCT <E>
  - <S> I/PRO like/V your/D hair/N ./PNCT <E>
  - <S> This/PRO is/V red/A ./PNCT <E>
- A: Transition Probability: $P(T_2|T_1) = C(\langle T_1, T_2 \rangle)/C(T_1)$
- A(<S>,D):
  - $P(D| < S >) = 1/3$
- A(<S>,PRO):
  - $P(PRO| < S >) = 2/3$

Final Calculation

- π: Initial Probability Distribution

Final Calculation

- π: Initial Probability Distribution: None

Final Calculation

- π: Initial Probability Distribution: None
- Sentence to predict: *This is beautiful*.

Final Calculation

- π: Initial Probability Distribution: None
- Sentence to predict: *This is beautiful*.
- $<S>$ *This* ...

Final Calculation

- π: Initial Probability Distribution: None
- Sentence to predict: *This is beautiful*.
- $<S>$ *This* ...
  - $P(This|D) * P(D|<S>) =$
  - $P(This|PRO) * P(PRO|<S>) =$

Final Calculation

- π: Initial Probability Distribution: None
- Sentence to predict: *This is beautiful*.
- $<S>$ *This* ...
  - $P(This|D) * P(D|<S>) = 1/2 * 1/3 = 1/6$
  - $P(This|PRO) * P(PRO|<S>) =$

Final Calculation

- π: Initial Probability Distribution: None
- Sentence to predict: *This is beautiful*.
- $<S>$ *This* ...
  - $P(This|D) * P(D|<S>) = 1/2 * 1/3 = 1/6$
  - $P(This|PRO) * P(PRO|<S>) = 1/2 * 2/3 = 2/6 = 1/3$

Final Calculation

- π: Initial Probability Distribution: None
- Sentence to predict: *This is beautiful*.
- $<S>$ *This* ...
  - $P(This|D) * P(D|<S>) = 1/2 * 1/3 = 1/6$
  - $P(This|PRO) * P(PRO|<S>) = 1/2 * 2/3 = 2/6 = 1/3$
- ... *This is* ...

Final Calculation

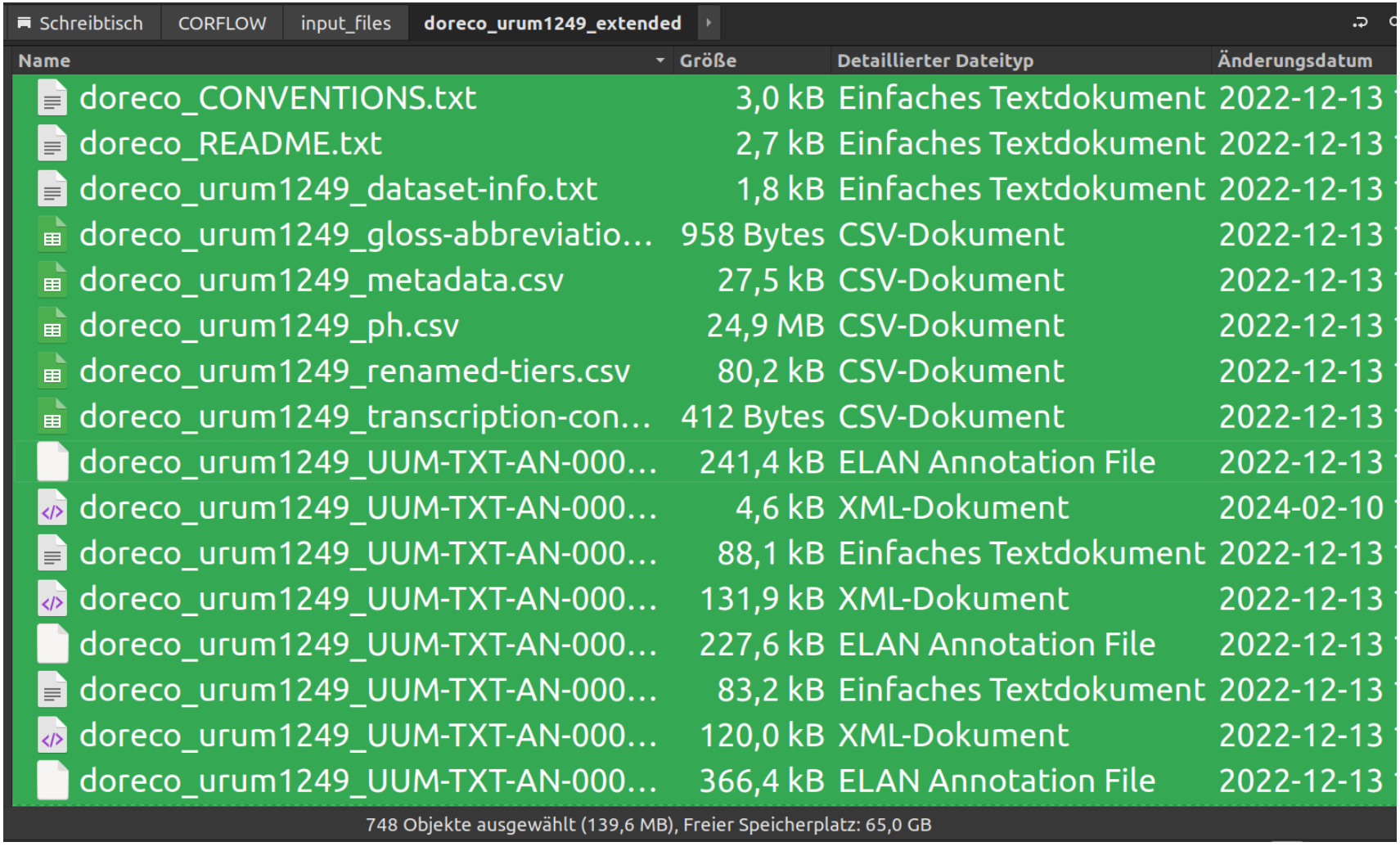
- π: Initial Probability Distribution: None
- Sentence to predict: *This is beautiful*.
- $<S>$ *This* ...
  - $P(This|D) * P(D|<S>) = 1/2 * 1/3 = 1/6$
  - $P(This|PRO) * P(PRO|<S>) = 1/2 * 2/3 = 2/6 = 1/3$
- ... *This is* ...
  - $\mathbf{1/3} * P(is|V) * P(V|PRO) = 2/3 * 2/2 = 2/9$

Final Calculation

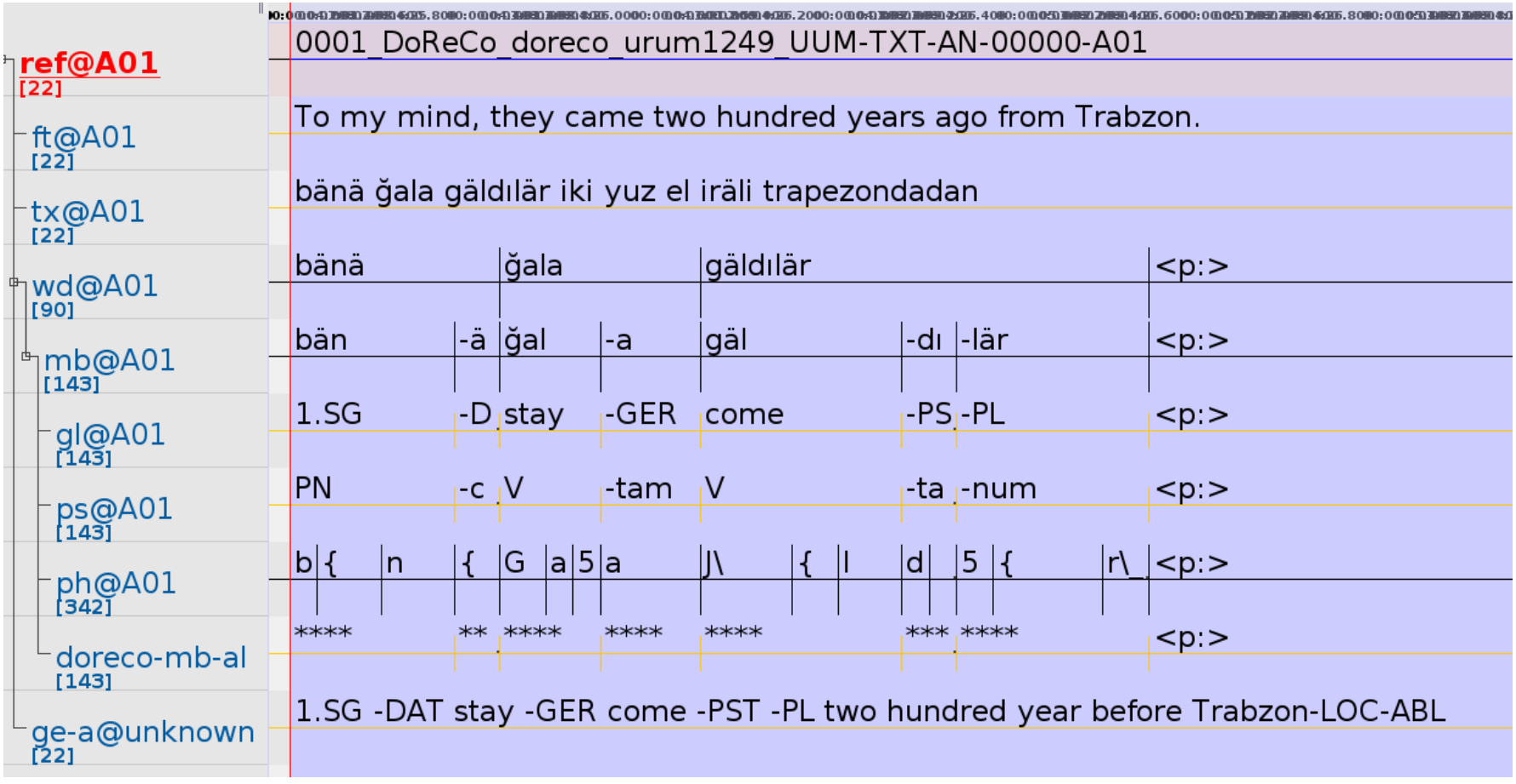
- π: Initial Probability Distribution: None
- Sentence to predict: *This is beautiful*.
- $<S>$ *This* ...
  - $P(This|D) * P(D|<S>) = 1/2 * 1/3 = 1/6$
  - $P(This|PRO) * P(PRO|<S>) = 1/2 * 2/3 = 2/6 = 1/3$
- ... *This is* ...
  - $\mathbf{1/3} * P(is|V) * P(V|PRO) = 2/3 * 2/2 = 2/9$
- ... *is beautiful* ...
  - $\mathbf{2/9} * P(beautiful|A) * P(A|V) = 1/2 * 2/2 = 2/18 = 1/9$
- ... *beautiful* . ...
  - $\mathbf{1/9} * P(.|PNCT) * P(PNCT|A) = 3/3 * 2/3 = 2/27$
- .... $<E>$
  - $\mathbf{2/27} * P(<E>|PNCT) = 3/3 = 2/27 = 0,074$

Outline

1. Download eaf Files



Tiers and Segments in ELAN



2. Get Word-Tag and Morph-Tag Pairs

scripts > get_urum_data_words.py > ...

```
5   from corflow import fromElan
6   import glob
7   import re
```

```
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
```

2. Get Word-Tag and Morph-Tag Pairs

```
38      #Acceptable doreco lables:
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
```

2. Get Word-Tag and Morph-Tag Pairs

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

```
21187   <E>;<E>
21188   čaimız;N
21189   var;V
21190   yazın;N
21191   gidirıh;V
21192   balıh;N
21193   dutmaya;V
21194   atdıhat;N
21195   edirıh;V
21196   <E>;<E>
21197
```

3. Import and Prepare the Data

```
231  #ACTUAL OPERATIONS BEGIN HERE#
232  data_path = "../data/"
233  tagged_words_file = "urum_tagged_words.txt"
234
235  #Get the word-tag-pairs from the respective file as tuples inside a list.
236  tagged_words_l = get_csv_as_list(data_path+tagged_words_file,";")
237  print(f"Loading data: {tagged_words_file}")
238
239  #Put word-tag-pairs into lists representing sentences and put them in one list.
240  tagged_sentences = get_sent_from_wd_tag_pairs(tagged_words_l,"<E>","<E>")
241  print(f"Creating sentences.")
```

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

```
[('bizım', 'PN'), ('halh', 'N'), ('čoğ', 'A'), ('išlier', 'V'), ('<E>', '<E>'), ('gürjülär', 'N'), ('sävme
rlär', 'V'), ('išlämäh', 'V'), ('onnar', 'PN'), ('emäh', 'N'), ('išlär', 'N'), ('säverlär', 'V'), ('<E>',
'<E>'), ('bizım', 'PN'), ('halh', 'N'), ('čoğ', 'A'), ('išlier', 'V'), ('yahši', 'A'), ('halhtır', 'N'), (
'<E>', '<E>'), ('nä', 'PN'), ('diem', 'V'), ('čto', 'PN'), ('ešyo', 'A'), ('<E>', '<E>'), ('güzäl', 'A'),
```

3. Import and Prepare the Data

```
21  def get_sent_from_wd_tag_pairs(wd_tag_pairs,wd_end,tag_end):
22      '''Returns a list containing sentences, where a sentence is represented as a list
        containing any number of word-tag-pairs (as tuples) from a given list of word-tag-pairs
        *wd_tag_pairs* (as tuples) except for those pairs, whose word equals *wd_end* and whose
        tag equals *tag_end*. These pairs are only used to encode the end of a sentence and the
        beginning of a following sentence in *wd_tag_pairs*.'''
23      tagged_sentences = []
24      single_sentence = []
25      for wd,tag in wd_tag_pairs:
26          if (wd != wd_end) & (tag != tag_end):
27              single_sentence.append((wd,tag))
28          elif single_sentence:
29              tagged_sentences.append(single_sentence)
30              single_sentence = []
31      return tagged_sentences
```

```
[[('bizım', 'PN'), ('halh', 'N'), ('čoğ', 'A'), ('išlier', 'V')], [('gürjülär', 'N'), ('sävmerlär', 'V'),
('išlämäh', 'V'), ('onnar', 'PN'), ('emäh', 'N'), ('išlär', 'N'), ('säverlär', 'V')], [('bizım', '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'), ('bizım', 'PN'), ('halh',
```

3. Split the Data into a Training and Test Set

```
sent_acc = 0
word_acc = 0
phases = 20

for i in range(phases):
    print(f"Phase: {i+1}/{phases}")

    #Create the training and test data randomly (optionally with a seed) by default in the ration 8:2 (train:test) from the list of tagged sentences.
    train_sentences,test_sentences = split_train_test(tagged_sentences)#,seed=469283984701)
    #Default seed I chose for comparing results: 469283984701

    #Apply the hmm algorithm and save the results.
    sent_pred,word_pred = viterbi_hmm_algorithm(train_sentences,test_sentences)
    sent_acc += sent_pred
    word_acc += word_pred

print(f"\nSentence Predition Accuracy: {round((sent_acc/phases)*100,2)}%.")
print(f"Word Prediction Accuracy: {round((word_acc/phases)*100,2)}%.")
```

3. Split the Data into a Training and Test Set

```
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)
```

4. Train the HMM

```
138  def viterbi_hmm_algorithm(train_sentences,test_sentences):
139      '''Creates an hmm model of *train_sentences* and predicts *test_sentences* based on that
         model. Returns a tuple with its first value being the accuracy of the hmm model when
         predicting sentences and its second value being the accuracy of the hmm model when
         predicting single (pos) tags.'''
140      print(f"Training.")
141      trans_prob_df = get_transition_probability(train_sentences)
142      test_sents = test_sentences
143      train_data = reduce_dim(train_sentences)
144      emis_prob_df = get_emission_probability(train_data)
```

4. Train the HMM: Emission Probability

```python
def get_emission_probability(train_data):
    '''Returns a pandas data frame representing the emission probability of all unique
    word-tag-pairs of a given list of word-tag-pairs *train_data*. In the final data frame,
    the rows represent the words and the columns the (pos) tags.'''

    def get_emission_probability_base(word:str,tag:str,data):
        '''Returns for calculating the emission probability of a *word* given a *tag* from a
        list of word-tag-pairs (tuples) *data* a tuple with two ints.'''
        #P(word|tag) = (P(word) n P(tag))/P(tag)
        #P(word|tag) = C(<tag,word>)/C(<tag,X>)
        pairs_with_tag = [(wd,t) for wd,t in data if t == tag]
        tag_count = len(pairs_with_tag)
        word_tag_count = len([wd for wd,t in pairs_with_tag if wd == word])
        return word_tag_count/tag_count

    unique_words = tuple(set([wd for wd,tag in train_data]))
    unique_tags = list(set([tag for wd,tag in train_data]))
    emis_prob_array = np.zeros((len(unique_words),len(unique_tags)))
    for wd_ind,wd in enumerate(unique_words):
        relevant_unique_tags = tuple(set([tag for word,tag in train_data if word == wd]))
        for tag in relevant_unique_tags:
            tag_ind = unique_tags.index(tag)
            emis_prob_array[wd_ind,tag_ind] = get_emission_probability_base(wd,tag,train_data)
    return pd.DataFrame(emis_prob_array,index=unique_words,columns=unique_tags)
```

4. Train the HMM: Emission Probability Matrix

	PN	NV	PRT	N	C/A	INF	INT	...	N-num	A/C/PN	INTJ	A	AN	REF.PN	PR
dägıštırdılar	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0
krımä	0.0	0.0	0.0	0.000225	0.000000	0.0	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0
ambelädä	0.0	0.0	0.0	0.000000	0.093023	0.0	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0
musulman	0.0	0.0	0.0	0.000450	0.000000	0.0	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0
gürjülärım	0.0	0.0	0.0	0.000225	0.000000	0.0	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
yapi	0.0	0.0	0.0	0.001573	0.000000	0.0	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0
ist	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0
izvestni	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	...	0.0	0.0	0.0	0.000366	0.0	0.0	0.0
heirli_olsun	0.0	0.0	0.0	0.000225	0.000000	0.0	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0
iz	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0

[4664 rows x 22 columns]

4. Train the HMM: Transition Probability

```
83  def get_transition_probability(training_sentences):
86      def get_transition_probability_base(tag1,tag2,data):
89          tags = [tag for word,tag in data]
90          count_tag1_tag2 = 0
91          for ind in range(len(tags)-1):
92              if (tags[ind] == tag1) & (tags[ind+1] == tag2):
93                  count_tag1_tag2 += 1
94          count_tag1_x = len([tag for tag in tags if tag == tag1])
95          return (count_tag1_tag2,count_tag1_x)
96
97      new_sentences = [sent.copy() for sent in training_sentences]
98      for sent in new_sentences:
99          sent.insert(0,("<S>","<S>"))
100         sent.append(("<E>","<E>"))
101     new_data = reduce_dim(new_sentences)
102     unique_tags = list(set([tag for wd,tag in new_data]))
103     trans_prob_array = np.zeros((len(unique_tags),len(unique_tags)))
104     for ind1,t1 in enumerate(unique_tags):
105         for ind2,t2 in enumerate(unique_tags):
106             if t1 == "<E>":
107                 trans_prob_array[ind1,ind2] = 0
108             elif t2 == "<S>":
109                 trans_prob_array[ind1,ind2] = 0
110             else:
111                 tag1_tag2,tag1_x = get_transition_probability_base(t1,t2,new_data)
112                 trans_prob_array[ind1,ind2] = tag1_tag2/tag1_x
113     return pd.DataFrame(trans_prob_array,columns=unique_tags,index=unique_tags)
```

4. Train the HMM: Transition Probability Matrix

	PN	NV	PRT	N	C/A	...	A	AN	<S>	REF.PN	PR
PN	0.091932	0.000000	0.006567	0.386023	0.000469	...	0.166041	0.0	0.0	0.000000	0.001407
NV	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.0	0.0	0.000000	0.000000
PRT	0.222581	0.000000	0.032258	0.180645	0.022581	...	0.187097	0.0	0.0	0.000000	0.000000
N	0.093055	0.000225	0.012812	0.217352	0.001349	...	0.125871	0.0	0.0	0.000000	0.000450
C/A	0.139535	0.000000	0.046512	0.302326	0.000000	...	0.116279	0.0	0.0	0.000000	0.000000
INF	0.000000	0.000000	0.000000	0.250000	0.000000	...	0.000000	0.0	0.0	0.000000	0.000000
<E>	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.0	0.0	0.000000	0.000000
INT	0.000000	0.000000	0.000000	0.400000	0.000000	...	0.200000	0.0	0.0	0.000000	0.000000
V	0.129616	0.000000	0.020074	0.226766	0.000496	...	0.157125	0.0	0.0	0.000000	0.001239
Pn.	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	1.0	0.0	0.000000	0.000000
Q\t	0.000000	0.000000	0.000000	1.000000	0.000000	...	0.000000	0.0	0.0	0.000000	0.000000
Q	0.044905	0.000000	0.001727	0.452504	0.001727	...	0.094991	0.0	0.0	0.000000	0.000000
PN-case	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.0	0.0	0.000000	0.000000
Pn.A	0.000000	0.000000	0.100000	0.500000	0.000000	...	0.200000	0.0	0.0	0.000000	0.000000
P	0.340426	0.000000	0.021277	0.287234	0.000000	...	0.202128	0.0	0.0	0.010638	0.000000
C	0.228178	0.000000	0.013783	0.202144	0.006126	...	0.176110	0.0	0.0	0.000000	0.007657
N-num	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.0	0.0	0.000000	0.000000
A/C/PN	0.082645	0.000000	0.016529	0.231405	0.000000	...	0.107438	0.0	0.0	0.000000	0.000000
INTJ	0.000000	0.000000	0.000000	1.000000	0.000000	...	0.000000	0.0	0.0	0.000000	0.000000
A	0.081715	0.000000	0.019421	0.293514	0.001466	...	0.185049	0.0	0.0	0.000000	0.000000
AN	0.000000	0.000000	0.000000	0.000000	0.500000	...	0.000000	0.0	0.0	0.000000	0.000000
<S>	0.271654	0.000000	0.043870	0.229471	0.009561	...	0.227784	0.0	0.0	0.000000	0.001687
REF.PN	0.000000	0.000000	0.000000	0.000000	0.000000	...	1.000000	0.0	0.0	0.000000	0.000000
PR	0.055556	0.000000	0.000000	0.388889	0.000000	...	0.111111	0.0	0.0	0.000000	0.000000

[24 rows x 24 columns]

5. Test the HMM: Viterbi Algorithm

```
145     print(f"Predicting.")
146     #Iterating over every tag, even if a word does not appear with that tag at all, seems to be
        slower than first getting only the relevant tags and only then iterating through those.
147     sentences_correctly_predicted = 0
148     words_correctly_predicted = 0
149     skipped_sents = 0
150     most_common_tag = get_most_common_tag(train_data)
151     for c,sent in enumerate(test_sents):
152         #print(f"{c+1}/{len(test_sents)}: {sent}")
153         paths = []
154         for ind,wd_tag_pair in enumerate(sent):
155
156             #'''
157             if len(paths) > 200_000:
158                 skipped_sents += 1
159                 print(f"Skip sentence: Too many calculated paths. At least {len(paths)}.")
160                 break
161             #'''
162
163             word = wd_tag_pair[0]
164             current_paths = []
165             tags = get_tag_types(word,train_data)
```

5. Test the HMM: Viterbi Algorithm

```
167             if ind == 0:
168                 if not tags:
169                     current_paths.append(([most_common_tag],1))
170                 for tag in tags:
171                     trans_p = trans_prob_df.at["<S>",tag]
172                     emis_p = emis_prob_df.at[word,tag]
173                     current_p = trans_p * emis_p
174                     if current_p == 0:
175                         continue
176                     current_tags = [tag]
177                     current_paths.append((current_tags,current_p))
```

```
178             else:
179                 if not tags:
180                     for tags_l,old_p in paths:
181                         current_tags = tags_l + [most_common_tag]
182                         current_paths.append((current_tags,old_p))
183                 for tag in tags:
184                     #Previously, I added emis_p inside the next loop, slowing
                        extremly. It only has to be calculated once per tag.
185                     emis_p = emis_prob_df.at[word,tag]
186                     for tags_l,old_p in paths:
187                         trans_p = trans_prob_df.at[tags_l[-1],tag]
188                         current_tags = tags_l + [tag]
189                         current_p = old_p * emis_p * trans_p
190                         if current_p == 0:
191                             continue
192                         current_paths.append((current_tags,current_p))
```

5. Test the HMM: Calculation Example

```
3/444: [('anam', 'N'), ('oldi', 'V'), ('jinistä', 'N'), ('ğırh', 'Q'), ('alti', 'Q'), ('elda', 'N')]
current paths: [(['N'], 0.0006645404737334224), (['A'], 8.073962569785879e-05)]
paths for the next word: [(['N'], 0.0006645404737334224)]
current paths: [(['N', 'V'], 3.1890381148618677e-06)]
paths for the next word: [(['N', 'V'], 3.1890381148618677e-06)]
word 'jinistä' not in training data.
current paths: [(['N', 'V', 'N'], 3.1890381148618677e-06)]
paths for the next word: [(['N', 'V', 'N'], 3.1890381148618677e-06)]
current paths: [(['N', 'V', 'N', 'Q'], 1.5512120377232247e-09)]
paths for the next word: [(['N', 'V', 'N', 'Q'], 1.5512120377232247e-09)]
current paths: [(['N', 'V', 'N', 'Q', 'Q'], 2.6600738673374235e-12)]
paths for the next word: [(['N', 'V', 'N', 'Q', 'Q'], 2.6600738673374235e-12)]
current paths: [(['N', 'V', 'N', 'Q', 'Q', 'N'], 1.7585457065337037e-15)]
paths for the next word: [(['N', 'V', 'N', 'Q', 'Q', 'N'], 1.7585457065337037e-15)]
final paths: [(['N', 'V', 'N', 'Q', 'Q', 'N'], 1.7585457065337037e-15)]
optimal path: (['N', 'V', 'N', 'Q', 'Q', 'N'], 1.7585457065337037e-15)
actual path: ['N', 'V', 'N', 'Q', 'Q', 'N']
```

5. Test the HMM: Viterbi Algorithm

```
193                paths = current_paths
194                if not paths:
195                    break
196                max_p = max(paths,key=itemgetter(1))[1]
197                paths = [path for path in paths if path[1] == max_p]
198
199            if not paths:
200                continue
201
202            for tags_l,old_p in paths:
203                trans_p = trans_prob_df.at[tags_l[-1],"<E>"]
204                if trans_p == 0:
205                    continue
206                old_p *= trans_p
```

5. Test the HMM: Viterbi Algorithm

```
208         max_p = max(paths,key=itemgetter(1))[1]
209         paths = [path for path in paths if path[1] == max_p]
210         random_ind = np.random.randint(0,len(paths))
211         optimal_path = paths[random_ind]
212         actual_path = [tag for wd,tag in sent]
213         #Check, whether the predictions are correct or not.
214         if optimal_path[0] == actual_path:
215             sentences_correctly_predicted += 1
216             words_correctly_predicted += len(actual_path)
217         else:
218             for index in range(len(optimal_path[0])):
219                 if optimal_path[0][index] == actual_path[index]:
220                     words_correctly_predicted += 1
221
222     sent_pred_accuracy = sentences_correctly_predicted/len(test_sentences)
223     word_pred_accuray = words_correctly_predicted/len(reduce_dim(test_sentences))
224
225     print(f"Skipped sentences due to too many calculated paths: {skipped_sents}/{len(test_sents)}")
226
227     return sent_pred_accuracy,word_pred_accuray
```

```
Phase: 19/20
Training.
Predicting.
Skipped sentences due to too many calculated paths: 0/444
Phase: 20/20
Training.
Predicting.
Skipped sentences due to too many calculated paths: 0/444

Sentence Predition Accuracy: 39.05%.
Word Prediction Accuracy: 85.32%.
```

Outline

All Datasets: Urum

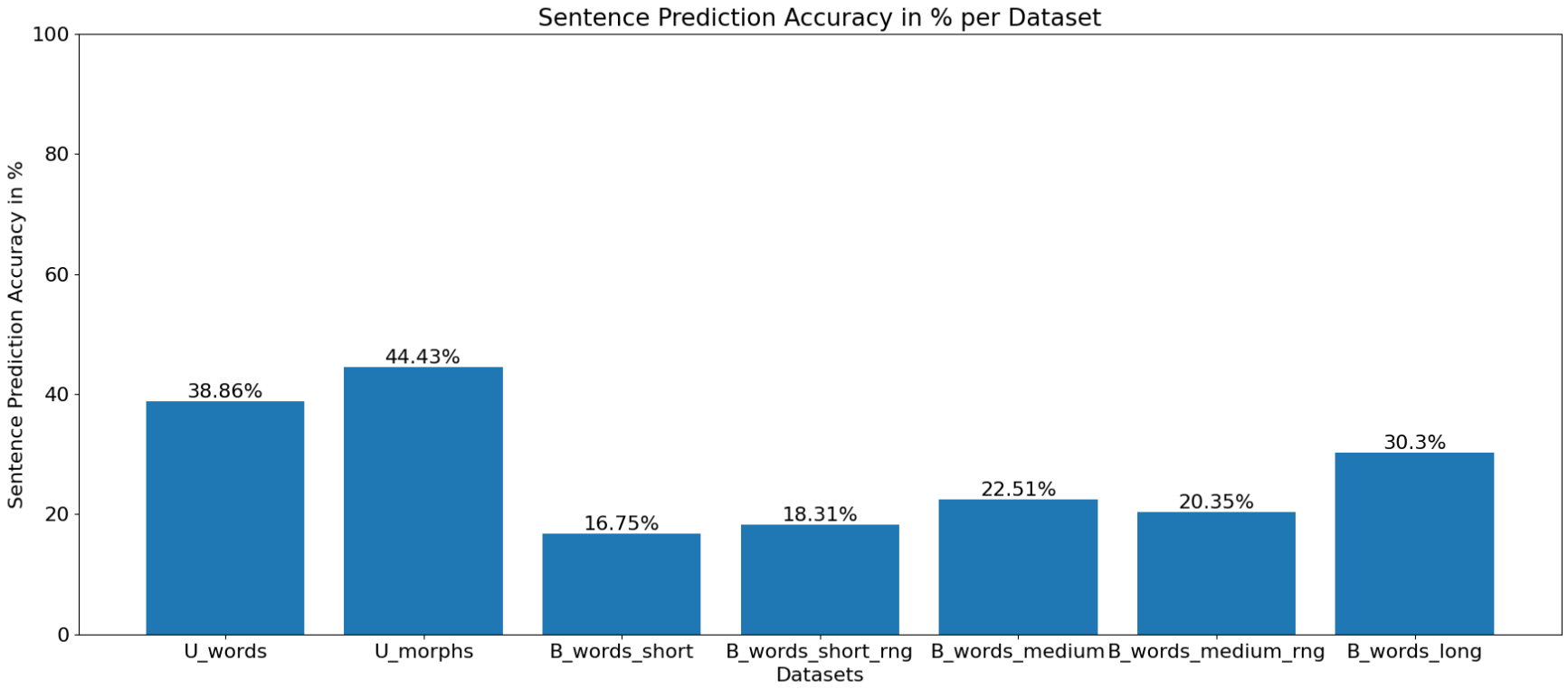
- urum_words:
  - Sentences: 2222
  - Tokens: 21192
  - POS-Tags: 25
- urum_morphs:
  - Sentences: 2212
  - Tokens: 36242
  - POS-Tags: 59

All Datasets: Brown

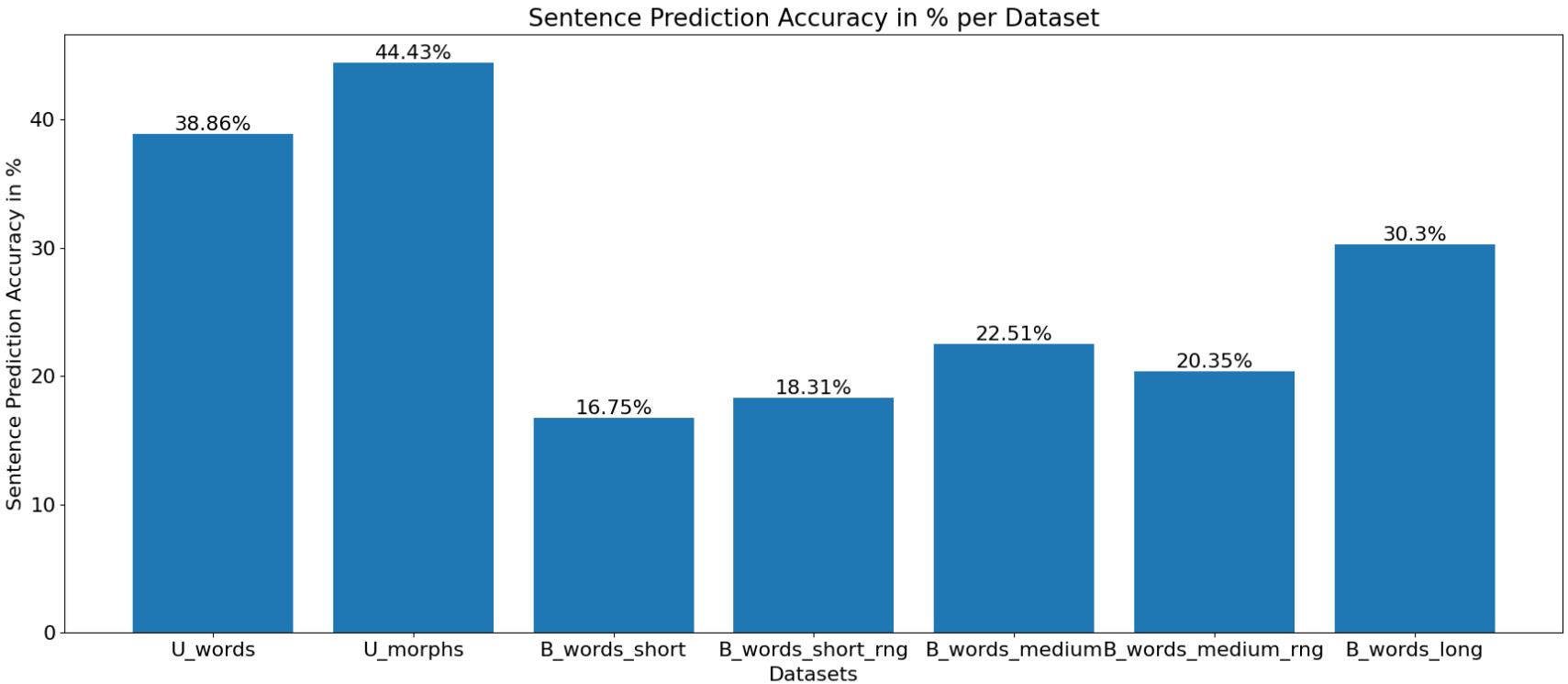
POS-Tags: 13

- brown_short:
  - Sentences: 917
  - Tokens: 21202
- brown_short_rng:
  - Sentences: 933
  - Tokens: 21234
- brown_medium:
  - Sentences: 1608
  - Tokens: 36263
- brown_medium_rng:
  - Sentences: 1629
  - Tokens: 36270
- brown_long:
  - Sentences: 4623
  - Tokens: 105176

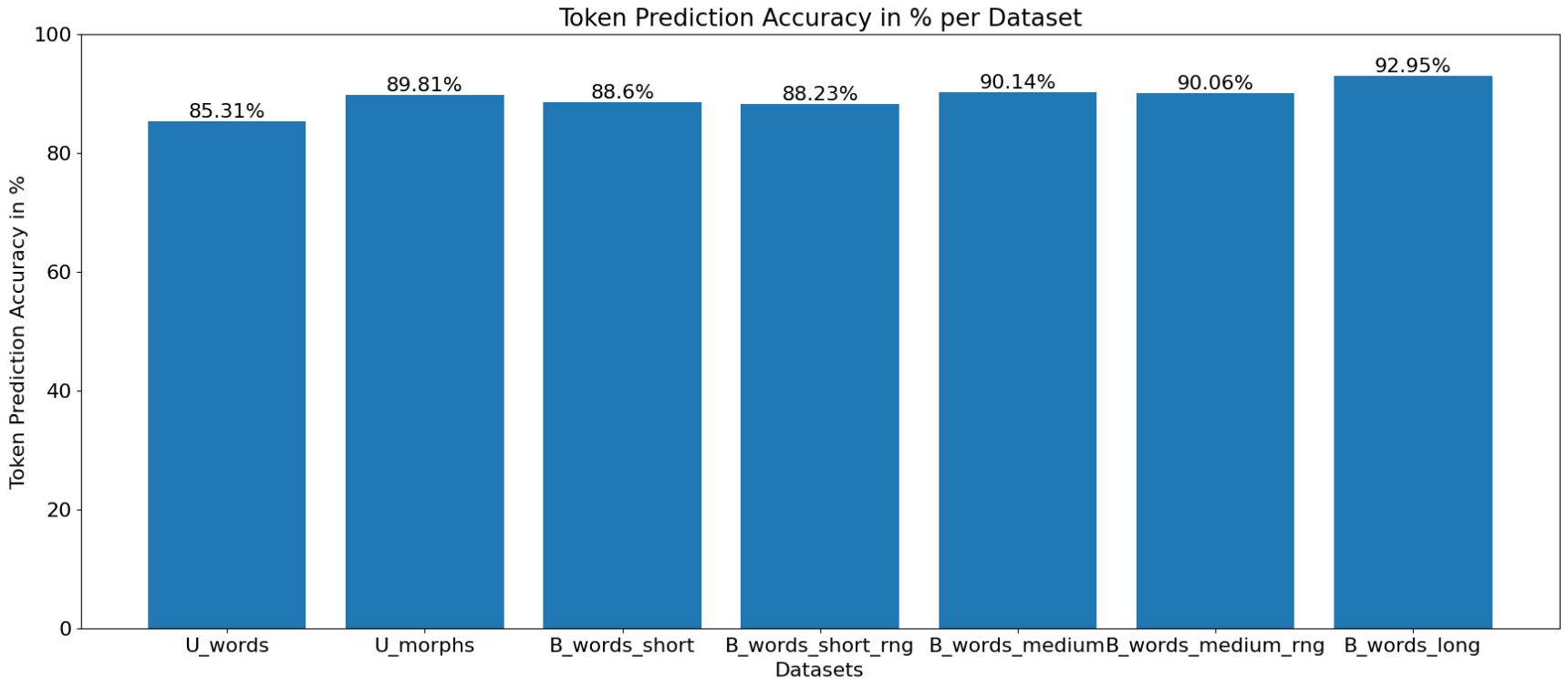
Results: Correctly Predicting all Tags in a Sentence



Results: Correctly Predicting all Tags in a Sentence



Results: Correctly Predicted Tags In Total



References

- 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/> (Accessed on 26/05/2024).
- Skopeteas, Stavros (2015): Caucasian Urums and Urum language.
- Seifart, Frank, Ludger Paschen & Matthew Stave (eds.). 2022. Language Documentation Reference Corpus (DoReCo) 1.2. Berlin & Lyon: Leibniz-Zentrum Allgemeine Sprachwissenschaft & laboratoire Dynamique Du Langage (UMR5596, CNRS & Université Lyon 2). DOI:10.34847/nkl.7cbfq779

References

- Skopeteas, Stavros, Violeta Moisidi, Nutsa Tsetereli, Johanna Lorenz and Stefanie Schröter. 2022. Urum DoReCo dataset. In Seifart, Frank, Ludger Paschen and Matthew Stave (eds.). Language Documentation Reference Corpus (DoReCo) 1.2. Berlin & Lyon: Leibniz-Zentrum Allgemeine Sprachwissenschaft & laboratoire Dynamique Du Langage (UMR5596, CNRS & Université Lyon 2). https://doreco.huma-num.fr/languages/urum1249 (Accessed on 26/05/2024). DOI:10.34847/nkl.ac166n10

References

- Paschen, Ludger, François Delafontaine, Christoph Draxler, Susanne Fuchs, Matthew Stave & Frank Seifart. 2020. Building a Time-Aligned Cross-Linguistic Reference Corpus from Language Documentation Data (DoReCo). In Proceedings of The 12th Language Resources and Evaluation Conference, 2657–2666. Marseille, France: European Language Resources Association. https://www.aclweb.org/anthology/2020.lrec-1.324 (2024/05/26).