# Tulajdonnév-felismerés német nyelven

# Általánosan a feladatról

- A feladatnak két fő része van:
  - megtalálni a tulajdonnevet a szövegben (ami az esetek többségében egy token egy sorban formában van)
  - ennek besorolása az előre meghatározott kategóriákba (Person, Location, Organization, Other)
- Szükség van egy minél nagyobb kézzel betaggelt szövegre, amelyen később tudunk tanítani és tesztelni
- A modern megközelítések neurális hálóval dolgoznak

# Kihívások

- Egymásba ágyazott tulajdonnevek megtalálása, amennyiben az annotáció erre lehetőséget ad egyáltalán
- Megfelelő korpusz megtalálása:
  - elég nagy méret
  - a területünkhöz illő téma
  - nincs tele hibákkal a címkézés
- Az emberi beszéd struktúráit megtanítani a gépnek (jelentés, mondatfelépítés)
- Mik is pontosan a tulajdonnevek (England – englander)

# GermEval2014

- A tanítás és a tesztelés a 2014-es GermEval NER Shared task adaton történt, ami részben a német wikipédiáról származik, és több mint 590000 tokent tartalmaz
- A német wikipédiáról és különböző híroldalakról lescrapelt szöveg

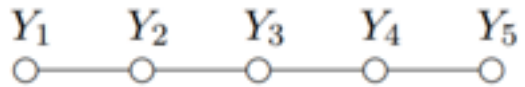
1 Aufgrund O O
2 seiner O O
3 Initiative O O
4 fand O O
5 2001/2002 O O
6 in O O
7 Stuttgart B-LOC O
8 , O O
9 Braunschweig B-LOC O

# Stanford NER modell

- Elérhető számos nyelven, legjobb eredmények angolul, de jelentős eredmények német, spanyol és kínai nyelven is
- Conditional Random Field (CRF) egy általános implementálása tulajdonképpen (CRFClassifier-nek is nevezik a modellt)
- Számos feature extractor is tartozik hozzá, amelyek tetszés szerint állíthatók a tanításhoz
- Letöltés után egyszerű parancsokkal lehet szöveget tokenizálni, taggelni és tanítani
- Demo: http://corenlp.run/

# Conditional Random Fields

- Egyfajta irányítatlan valószínűségi gráfmodell
- Olyan osztályozó ami figyelembe veszi egy adott elem környezetét is a jóslatnál
- NLP-ben a Linear Chain CRF-ek használatosak, így van ez a használt modellnél is




- Ez valójában egy Hidden Markov Modelnek felel meg

# HMM

- A HMM természetesen rendelkezik a prediktív modellezésben igen hasznos Markov tulajdonsággal
- Definíció szerint adott két diszkrét sztochasztikus folyamat, X_n és Y_n (n>=1), ekkor az (X_n, Y_n) pár Hidden Markov Model, ha:
  - X_n Markov-folyamat és nem közvetlenül megfigyelhető (innen ered a név)
  - $\mathbf{P}\left(Y_n \in A \,\middle|\, X_1 = x_1, \ldots, X_n = x_n\right) = \mathbf{P}\left(Y_n \in A \,\middle|\, X_n = x_n\right)$
- minden n>=1, x_1, … , x_n-re és egy tetszőleges mérhető A halmazra
- Tehát fontos még, hogy Y_n viselkedése függ az X_n-től
- Ezt kihasználva szeretnénk a feltételes valószínűségeket maximalizálni a paraméterek optimalizálásával

# Feature extractorok

- A tanítóadatból előre definiált függvények segítségével featureöket gyárt, amiken majd később lehet optimalizálni
- A modell felépítésének köszönhetően ezeket egyszerűen lehet ki-, bekapcsolni, illetve a paramétereiket állítani egy fájlban
- Néhány egyszerűbb példa:
  - useWord: maga a szó is egy feature lesz
  - useNGrams: ha True, akkor használ betű n-gramokat
  - useMidNGrams: használjon-e olyan karakter n-gramokat, ami nem tartalmazza a szó első és utolsó betűjét sem
- De itt kell megadni azt is például milyen formában van a tanítófájl (1. oszlop a token, 2. az osztály)

# Distributional Similarity Classes

- Megadható egy a szavak hasonlóságát szemléltető fájl is a modellnek
- Ebben minden szóhoz egy osztályt rendelünk, a nagyon hasonlóak értelemszerűen egy osztályba kerülnek, és minél különbözőbbek, annál távolabbiba
- Az alapértelmezett fájl nem volt elérhető
- Saját ún. DistSim Lexikont kellett készíteni, mivel a kívánt formában nagyon nehéz találni előre elkészítettet

# Wordembedding

- Itt jönnek képbe a szóvektorok, amelyekről ismert, hogy a szavakat elhelyezik egy n-dimenziós térben és így szintén mérhető a hasonlóságuk
- Ilyet már nem olyan nehéz találni, a fastText 2 millió szavas, előre betanított szóvektorai jó kiindulási alapot biztosítottak
- Ezek azonban 300 dimenziós vektorok, így a memóriával óvatosan kellett bánni
- Ezen felül a 2 millió szó is kicsit túlzásnak tűnhet a ~470 ezres tanítóadaton
- Kísérleteztünk dimenziócsökkentéssel a vektorokon Uniform Manifold Approximation and Projectionnel (UMAP), de végül az tűnt gazdaságosabbnak, ha a 2 millió szónak hagyjuk el a többségét
- Az optimálishoz legközelebbinek a y szó felhasználása bizonyult a 300 dimenziós vektorokkal, ezen végeztünk k-közép klaszterezést k=70-nel
- Az így kapott klaszterek képezik a kívánt Lexikont

# UMAP

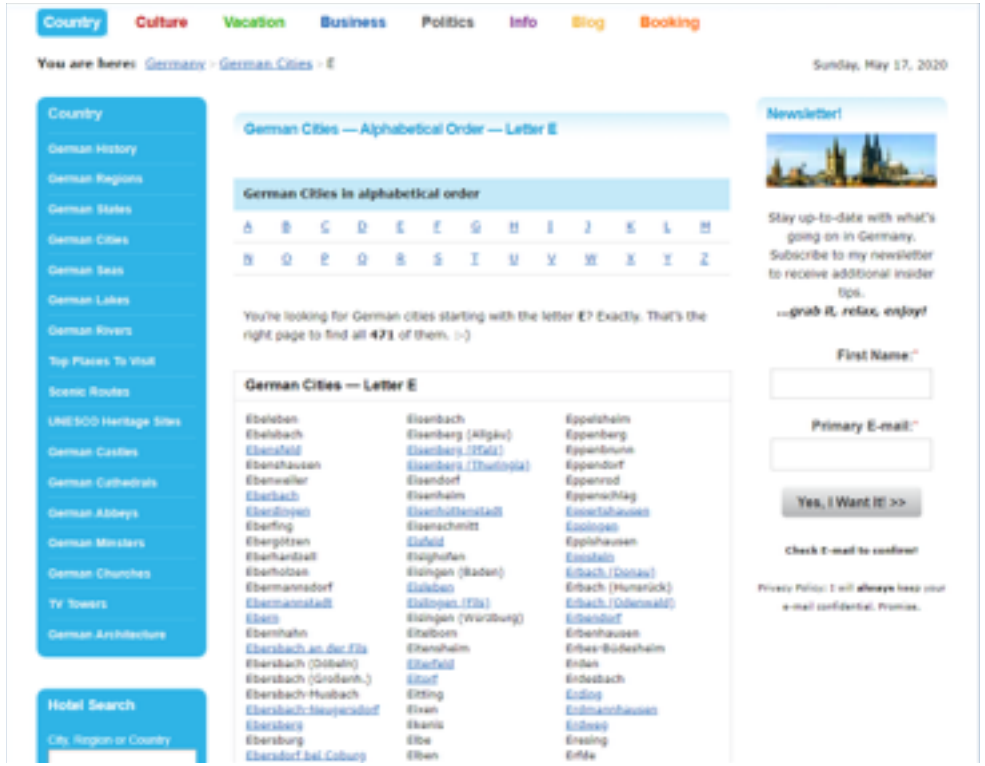
- Az algoritmus 3 feltevésen alapszik:
  - Az adat egyenletes eloszlású a Riemann-felületen;
  - A Riemann-metrika lokálisan konstans (vagy legalább lehet közelíteni);
  - A felület lokálisan összefüggő;
- Így modellezhető az adat fuzzy topológikus struktúrával, és megtalálható egy kisebb dimenziós vetítés, ami leginkább hasonlít ehhez a struktúrához
- Könnyen használható python csomag, kifejezetten ajánlott klaszterezés előtti előfeldolgozási lépésként nagy dimenzió esetén
- Négy fő paraméter: szomszédok száma (globális/lokális), minimális távolság, komponensek száma, metrika
- Sajnos végül nem tudtuk kihasználni, ugyanis értelmes tokenszám mellett 300 vektorról a redukció túl sok memóriát használt

# Gazette feature

- Működése röviden: egy csak tulajdonneveket tartalmazó fájlt olvas be, a hozzájuk tartozó címkével együtt
- Akkor jön létre a feature, ha tiszta egyezés van a tanítás során
- A kezdeti modell hibáit vizsgálva adódott az ötlet (egész gyakori volt, hogy egy-két szavas helyneveket nem ismert fel)
- Egyáltalán nem szerepel eredetileg, de egy megírt extractor
- Ismét saját adatot kellett konstruálni

# Gazette feature

- Webscraping segítségével sikerült a helynevekre egy jó adatot generálni
- Sikerült javulást is elérni vele, még ha nem is hatalmasat




```
LOC Albershausen
LOC Albersweiler
LOC Albertshofen
LOC Albessen
LOC Albig
LOC Albisheim
LOC Albsfelde
LOC Albstadt
LOC Aldenhoven
LOC Aldersbach
LOC Aldingen
LOC Alerheim
LOC Alesheim
LOC Aletshausen
LOC Alf
LOC Alfdorf
LOC Alfeld
LOC Alfhausen
LOC Alflen
LOC Alfstedt
LOC Alfter
```

# Quasi-Newton optimizer

- A neurális háló paramétereinek optimalizálásért felel
- Nemlineáris programozásban használatos algoritmus, amikor a teljes Newton módszerek túl sok erőforrást igényelnének
- Kétszeresen differenciálható függvények minimumát találja meg, kevesebb számítással
- Menete:
  1. Kiválasztunk egy x_0 kezdőpontot
  2. Approximáljuk a Hesse-mátrix inverzét, ezzel kapjuk meg, melyik irányban keressük az optimumot (egy állítható paraméter segítségével megadható mennyire legyen pontos, vigyázva a memóriával)
  3. A $x^{k+1} = x^k - [H^{-1}]_k * grad(x^k)$ képlettel frissítjük x-et
  4. Ellenőrizzük a konvergenciát
  5. Ha nem vagyunk elégedettek, vissza a 2. pontra

# Kiértékelés

- True Positive, False Positive, False Negative jóslatok számontartásával történik
- A mértékek pedig az ebből számítható Precision, Recall és F-mérték
- Az eredeti adatnak csak az elsődleges címkéit használtuk fel, azokon is átalakításokat kellett végezni, hogy megkapjuk végül a 9 kívánt osztályt (O, B-PER, I-PER, stb.)
- Ezeken való kiértékelés után az eredmények a kezdeti modellen:

| Entity | P | R | F1 | TP | FP | FN |
|---|---|---|---|---|---|---|
| LOC | 0.8407 | 0.7390 | 0.7866 | 776 | 147 | 274 |
| ORG | 0.7316 | 0.6237 | 0.6734 | 368 | 135 | 222 |
| OTH | 0.6734 | 0.4422 | 0.5339 | 134 | 65 | 169 |
| PER | 0.8200 | 0.7291 | 0.7719 | 533 | 117 | 198 |
| Totals | 0.7960 | 0.6773 | 0.7319 | 1811 | 464 | 863 |

# Eredmények

- A német helyneveket tartalmazó fájlból generált gazette feature-ökkel:

| Entity | P | R | F1 | TP | FP | FN |
|---|---|---|---|---|---|---|
| LOC | 0.8486 | 0.7476 | 0.7949 | 785 | 140 | 265 |
| ORG | 0.7241 | 0.6271 | 0.6721 | 370 | 141 | 220 |
| OTH | 0.6734 | 0.4422 | 0.5339 | 134 | 65 | 169 |
| PER | 0.8193 | 0.7319 | 0.7731 | 535 | 118 | 196 |
| Totals | 0.7972 | 0.6821 | 0.7352 | 1824 | 464 | 850 |

- Szóvektorral kiegészítve:
  - 125000 szó, 30 dimenzió kivágva a 300ból, ez még lecsökkentve 5 dimenzióba és dimenziócsökkentés nélkül:

| Entity | P | R | F1 | TP | FP | FN |
|---|---|---|---|---|---|---|
| LOC | 0.8471 | 0.7705 | 0.8070 | 809 | 146 | 241 |
| ORG | 0.7162 | 0.6203 | 0.6649 | 366 | 145 | 224 |
| OTH | 0.7173 | 0.4521 | 0.5547 | 137 | 54 | 166 |
| PER | 0.8306 | 0.7715 | 0.8000 | 564 | 115 | 167 |
| Totals | 0.8031 | 0.7016 | 0.7489 | 1876 | 460 | 798 |

| Entity | P | R | F1 | TP | FP | FN |
|---|---|---|---|---|---|---|
| LOC | 0.8402 | 0.7610 | 0.7986 | 799 | 152 | 251 |
| ORG | 0.7054 | 0.6169 | 0.6582 | 364 | 152 | 226 |
| OTH | 0.7204 | 0.4422 | 0.5481 | 134 | 52 | 169 |
| PER | 0.8141 | 0.7729 | 0.7930 | 565 | 129 | 166 |
| Totals | 0.7934 | 0.6963 | 0.7417 | 1862 | 485 | 812 |

# Eredmények

- 125000 sor, 300 dimenziós vektorokkal futtatott k-középből kapott hasonlósági osztályokkal:

| Entity | P | R | F1 | TP | FP | FN |
|---|---|---|---|---|---|---|
| LOC | 0.8509 | 0.7829 | 0.8155 | 822 | 144 | 228 |
| ORG | 0.7431 | 0.6373 | 0.6861 | 376 | 130 | 214 |
| OTH | 0.7143 | 0.4620 | 0.5611 | 140 | 56 | 163 |
| PER | 0.8447 | 0.8112 | 0.8276 | 593 | 109 | 138 |
| Totals | 0.8148 | 0.7221 | 0.7657 | 1931 | 439 | 743 |

- Tehát az F-mértéken több, mint 3 százados javulás
- A GermEval versenyen elért legjobb eredmény 0.7908 (igaz ez valószínűleg saját modell és a teszthalmazon lett kiértékelve)

# Források

- Stanford NER modell
  - https://nlp.stanford.edu/software/CRF-NER.shtml
  - https://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/ie/NERFeatureFactory.html
- CRF
  - https://en.wikipedia.org/wiki/Conditional_random_field
  - https://dms.sztaki.hu/sites/dms.sztaki.hu/files/file/2013/crf_slide.pdf
- Quasi-Newton
  - https://optimization.mccormick.northwestern.edu/index.php/Quasi-Newton_methods
- HMM
  - https://pdfs.semanticscholar.org/9528/4b31f27b9b8901fdc18554603610ebbc2752.pdf
  - https://en.wikipedia.org/wiki/Hidden_Markov_model
- UMAP
  - https://umap-learn.readthedocs.io/en/latest/
- German cities
  - https://www.mygermancity.com/german-cities