Dokumentation POS-Tagger

Das Projekt ist ein Git-Repository, es gibt drei Branches. Diese sind jeweils nach der Aufgabe benannt.Das Projekt wurde mit Python 3.10 entwickelt. Es werden keine zusätzlichen Bibliotheken benötigt. Das Projekt besteht aus den Klassen hmm.py, classify.py und project.py.

Die Dokumentation im Python ist sehr ausführlich und gut dokumentiert. Dort wird auch nochmals auf Dateiformart usw. eingegangen. Im Branch „Aufgabe 3“ findet man die finale Dokumentation und den geschönten Code. Es sind auch nicht alle Modi in jeder Aufgabe vorhanden, aber immer mind. tag, train, eval. In den verschiedenen Branches sind auch die Ergebnis-Dateien eval.txt zeigt die Unterschiede im diff-Format. Alle anderen Dateien sind sinnvoll beschriftet.

Ich habe versucht den Funktionen und Variablen sinnvolle Namen zugeben, die Datentypen der Argumente und Rückgabe Typen wurden markiert. Fehler Fälle in der I/O werden versucht abzufangen. Es wird nur mit Log-Wahrscheinlichkeiten gearbeitet.

Es gibt verschiedene Modi, um das Programm zu nutzen. Um einen Modi zu nutzen wird dieser als zweites Argument in Python übergeben.

python3 project.py <Modi> <Argumente …>

Die Modi sind:

- train
    - Dieser Modus tanniert das HMM mithilfe eines schon getaggten Korpus. Das Resultat sind Bi-Gramme und die Emissions Wahrscheinlichkeiten (hidden states), welche jeweils in einer Text-Datei gespeichert werden (Format siehe unten). Falls die Datei vom Bi-Gramm oder der Emissions schon bestehen, werden diese überschrieben, wenn nicht wird diese neu erstellt.
    - Erwartet als Argument <training corpus> <bi gramm output> <emission output>
    - python3 project.py train tiger22/tiger22-train.tsv tiger22.bigrams tiger22.lex
    - Es gibt einen denoising Modus beim Trainieren, dies hat den Grund das die Daten zum Teil nicht korrekt getagt sind bzw. es da sicherlich unterschiedliche Möglichkeiten gibt. Der denoising erzwingt, dass jede Zahl, die mit Ziffern dargestellt wird, den Tag CARD zugewiesen bekommt, das gleiche passiert mit Links (XY) und zerschnitten Wörtern (TRUNC), dass impliziert nicht das es in jedem Fall korrekt ist. Der Name des Modus ist statt train, train-denoising.
- tag
    - Dieser Modus geniert mit Hilfe des HMM die Tags für einen nicht annotieren Text. Die Emissions- und Trans-Wahrscheinlichkeiten müssen vorher mit train berechnet worden sein.
    - Falls die Output Datei schon existiert, wird diese überschrieben.
    - Als Arguemtne warden erwarten: tag <bi gramm input> <emission input> <corpus> <tagged corpus out>
    - python3 project.py tag tiger22.bigrams tiger22.lex tiger22/tiger22-eval.txt tagged_text.tag
- eval
    - Dieser Modus vergleich zwei Text Datei, bestimmt das Delta draus und berechnet in % wie viele der Zeilen übereinstimmen.
    - Am Ende wird eine Statistik ausgegeben diese beinhaltet: Akkuratheit (in %), Anzahl Zeilen, Anzahl Korrekter Zeilen
    - Es wird eine Warnung ausgegeben, falls die Anzahl der beiden Dateien, die verglichen werden sollen nicht übereinstimmen.

    - Das Delta was bestimmt wird, wird in einer Datei gespeichert. Das Format ist ein „diff“ Format, man kann es aus der Python-Doku entnehmen.
    - https://de.wikipedia.org/wiki/Diff
    - Erwartet werden folgende Argumente: <vergleichsdatei> <eval-datei> <diff output>
    - python3 project.py eval tagged_text.tag tiger22/tiger22-eval.tsv eval.txt
- train-tag-eval
    - Er macht genau was er sagt, es ist die sequenzielle Ausführung der Modi: train, tag, eval. Man kann auch statt train-tag-eval den denoising Mode mit train-denoising-tag-eval anschalten.
    - train-tag-eval <training corpus> <bi gramm output> <emission output> <bi gramm input> <emission input> <corpus> <tagged corpus out> <vergleichsdatei> <eval-datei> <diff output>
    - python3 project.py train-tag-eval tiger22/tiger22-train.tsv tiger22.bigrams tiger22.lex tiger22.bigrams tiger22.lex tiger22/tiger22-eval.txt tagged_text.tag tagged_text.tag tiger22/tiger22-eval.tsv eval.txt

Format:

Die Bi-Gramme werden im folgenden Format gespeichert:

Tag 1 und Tag 2 stehen in folgender Beziehung zueinander P(Tag 2 | Tag 1)

<Wahrscheinlichkeit>\t<Tag 1>\t<Tag 2>\n

Beispiel:

-1.5616005830199404 START ART
-2.0077619123908983 START APPR

Die Datei endet immer mit \n

Emissionen:

<Wahrscheinlichkeit>\t<Tag 1>\t<Wort>\n

Beispiel:

-8.925719086722959 $('
-8.55046775721129 $(...
-12.545605669349944 $(^...
-11.69830780896274 NE ^Jaschke

Die Datei endet immer mit \n

Der getagte Text wird im Folgenden Format gespeichert, dass gleiche Format wird beim Trainieren erwartet.

<Wort>\t<Tag>\n
…
\n
Nächster Satz

Die Sätze werden mit einer leeren Zeile getrennt. Die Text Datei endet mit \n\n

Es wird ein kleiner Trick im Kodieren der Wörter verwendet, wenn beim Trainieren das Wort am Anfang des Satzes steht, wird dies mit ^ markiert und auch so in den Emissions-Wahrscheinlichkeiten gespeichert. In der Methode decode im HMM, wird der Input aufbereitet, sodass aus dem Satz [Das, Saarland, ist, 1, Kreis,.] aufbereitet folgendes wird: [^Das, Saarland, ist, NUMBER, Kreis,.]. Dies gilt auch für Zahlen, zerteilten Wörtern und Links (siehe weiter unten).

Aufgrund der HMM wird ein Ähnlicher Trick auch im Bezug der Bi-Gramme verwendet, sodass die Bigrame immer mit einem „Start“ Tag anfangen z.B. (START, ART) usw. HMM brauchen Start zustände.

```
"NUMBER", "LINK", "TRUNC",
"OVV3-CAPITAL", "OVV3-LOW", "OVV3-START", "OVV3",
"OVV6-CAPITAL", "OVV6-LOW", "OVV6-START", "OVV6",
"OVV9-CAPITAL", "OVV9-LOW", "OVV9-START", "OVV9",
"OVV-HUGE-CAPITAL", "OVV-HUGE-LOW", "OVV-HUGE-START", "OVV-HUGE"
```

Mein findet sämtliche Regulären Ausdrücke in der Datei: classify.py

Die Zahl nach OVV bedeutet die max. Länge, ein Wort kann nur max. in einer Kategorie sein, es gibt folgende OVV-Kategorien:

- CAPITAL
  - Erster Buchstabe des Wortes ist großgeschrieben
- LOW
  - durchgehend klein geschrieben.
- START
  - das Wort stand am Anfang des Satzes. Das heißt nicht, dass der erste Buchstabe großschrieben sein muss. Beim Trainieren werden Wörter, welche am Satzanfang mit ^ markiert. Im HMM wird das gleiche getan.
- OVV-3 usw.
  - Das sind alle Wörter, die nicht mit den vorherigen Regulären Ausdrücken übereinstimmen, das sind vor allem Fremdwörter, Kunstwörter oder Wörter mit alternativer/besondere Orthografie.
- NUMBER
  - Alle Zahlen auch Gleitkomma inkl. mit Komma oder Punkt getrennt.
- TRUNC
  - Wörter die mit einem Bindestrich enden aber mit mind. einem Buchstaben anfangen.
- LINK
  - Wörter die http anfangen für eine genaue Erklärung:
  - https://stackoverflow.com/questions/3809401/what-is-a-good-regular-expression-to-match-a-url

Die Ausdrücke haben zum Teil Fehler drin, das Wort „US-Start“ wird nicht als OVV-Capital klassifiziert, da ich davon ausgehe das nur der erste Buchstabe großgeschrieben wird, das gleiche Problem ist bei durchgehend großgeschrieben Worten. Alle diese Worte enden in der OVV-N Kategorie. Des Weiteren wird versucht jeden deutschen Umlaut korrekt zu behandeln, dies gilt aber nicht für Fremdwörter welche außerhalbe des deutschen Alphabet sind, aber dennoch im deutschen mit andren Buchstaben geschrieben werden.

Ich verwende in der Aufgabe 3 noch einen anderen Trick, es wird nicht daran entschieden, ob das Wort im Korpus drin vorkommt, sondern wie oft dieses Wort mit dem diesem Tag vorkommt. Wenn das Wort mit dem Tag nur einmal im Korpus vorkommt, wird es durch OVV ersetzt, dies hat zufolge dass OVV sich für das jeweilige Tag verschiebt anstatt wie in Aufgabe 2. Es wird aber nicht aus den Emissions-Daten gelöscht, es erhöht zusagen nur dass OVV des jeweiligen Tags ohne komplett zu verschwinden.

Ich habe mich direkt für ein großes Set an OVV Wörtern entschieden und wollte auch insbesondere den Start markieren. Die Grund warum die länge 3, 6, 9 und Huge jeweils ist, dass im deutsch bei der länge 3 sehr oft „grammatische Wörter“ wie Verben etc. dabei sind, aber selten Nomen, 6 ist eine länge welche viele Verben und Nomen haben. 9 ist insbesondere interessant da sehr oft Verben im Partizip sehr lang werden können, deswegen auch die Unterscheidung zwischen groß und klein. In die Huge Kategorie fallen sehr oft länge Nomen.

Dadurch dass es wirklich viele Klassen sind, kann er Fall eintreten, dass ein Wort als OVV-9-LOW markiert wird (im HMM), aber es nur einen Eintrag zur OVV9-CAPTIAL gibt um diese Problem zu lösen, gibt es die Methode „smooth_emissions“ in project.py (sehr gut beschrieben). Kurzzusammengefasst, versucht diese Methode diese Lücken zu füllen in dem man den Durchschnittswert der jeweiligen Klasse nimmt oder zu jeweiligen höheren und kleineren geht. Wenn es kein OVV gibt, resultiert dies in einem Fehler bzw. die Traningsdaten sind zu wenige. Bestimmte Klassen / Tags wie Artikel sind komplett ausgeschlossen davon, da diese normalerweise komplett bekannt sind. Die smooth Funktion führt zu keinem bessern Ergebnis.

Die Daten sind verrauscht z.B. wenn man es mit Link trainiert:

```
-12.950321602139203 XY LINK
-12.257174421579258 NN LINK
-11.158562132911149 NE LINK
```

Mit dem denosing Feature würde die Einträge zur NN und NE wegfallen und alle Einträge, unter XY fallen, dass gleiche gilt für NUMBER und TRUNC, diese sind auch „verrauscht“. Ohne das denosing Feature wird praktisch kein Link erkannt. Das denosing erhört nur die Wahrscheinlichkeit, wenn es nur mit der Klasse LINK verwendet wird, mit NUMBER und TRUNC führt es zu einer kleineren Wahrscheinlichkeit.

Man erkennt deutliche Verbesserungen, wenn man bestimmte Klassen von OVV ein oder ausschaltet:

Mit OVV-N-START: -0,002%

Mit NUMBER: +0,011%
Mit TRUNC: +0,067%
Mit LINK ohne denosing: +0%
Mit LINK mit denosing: +0,004%
Verschiedene Klassen: + 2,19%
Mit Smooth-Funktion: -0,01%

Es kann sein dass diese Daten zum Teil geringfügig abweichen, wenn man sie neu messen würde, da es zum Teil Änderungen gab bzw. Bug-Fixes aber dieser Faktor kann zum Teil sehr stark sein, aber auch nicht. Durch die impl meines „Tricks“ hat sich auch einiges verschoben.

```
Aufgabe 1
Akkuratheit: 21.60645453858363%
Total: 105848
Korrekte Wörter: 22870

Nur OVV (Aufgabe 2)
Akkuratheit: 92.31067190688535%
Total: 105848
Korrekte Wörter: 97709o

Denosing mit allen Klassen mit smooth
Akkuratheit: 94.45431184339807%
Total: 105848
Korrekte Wörter: 99978

Ohne NUMBER mit smooth
Akkuratheit: 94.33999697679691%
Total: 105848
Korrekte Wörter: 99857

Ohne NUMBER & TRUNC mit smooth
Akkuratheit: 94.2738644093417%
Total: 105848
Korrekte Wörter: 99787

Denosing nur für LINK
Ohne NUMBER & TRUNC
Mit smooth
Akkuratheit: 94.27764341319627%
Total: 105848
Korrekte Wörter: 99791

Ohne OVV-N-Start mit smooth
Akkuratheit: 94.50060464061674%
Total: 105848
Korrekte Wörter: 100027

mit allen Klassen ohne smooth
Akkuratheit: 94.73773713249187%
Total: 105848
Korrekte Wörter: 100278

ohne OVV-N-Start ohne smooth
Akkuratheit: 94.75852165369209%
Total: 105848
Korrekte Wörter: 100300
```

Die Ergebnisse von „ohne Smooth und Start“ liegen in einem extra Branch drin.