Übungsblatt 11

09.01.19

Präsenzaufgaben

Aufgabe 1 Herunterladen von Ressourcen

Laden Sie sich zunächst die Ressource `corpora/treebank` über den NLTK Download-Manager herunter.

```
1 import nltk
2 nltk.download()
```

Aufgabe 2 Von Daten zu Regelwahrscheinlichkeiten

Gegeben sei folgende kontextfreie Grammatik:

```
1  S → NP VP
2  VP → V NP PP
3  VP → V NP
4  NP → DET N
5  NP → NP PP
6  PP → P NP
7
8  DET → "the" | "a"
9  N → "boy" | "woman" | "telescope"
10 V → "saw"
11 P → "with"
```

Sie modelliert sehr einfache Sätze der Form `SBJ` *saw* `OBJ` mit optionaler Präpositionalphrase am Ende. Diese Präpositionalphrase kann entweder der näheren Bestimmung des Objekts oder der näheren Bestimmung der in der Verbalphrase ausgedrückten Handlung dienen.

Im folgenden sollen aus der Penn Treebank Wahrscheinlichkeiten für die einzelnen Regeln extrahiert werden, um dieser Ambiguität Herr zu werden.

(a) Nutzen das im NLTK enthaltene Sample der Penn Treebank (nach Installation unter `nltk.corpus.treebank` zu finden) zunächst zur Identifikation der für eine Disambiguierung nützlichen (Teil-)bäume der Penn Treebank.

Hinweis: Sie können sich bei der Analyse auf die jeweils 30 häufigsten Konstruktionen der Baumbank beschränken.

Anmerkung: Das Jupyter Notebook für diese Woche enthält nützliche Code-Templates dafür.

(b) Zählen Sie nun, wie oft die jeweiligen Konstruktionen in der Penn Treebank vorkommen und berechnen Sie die relativen Häufigkeiten als Approximation der Regelwahrscheinlichkeiten. Das Vorgehen wird in folgender Formel veranschaulicht:

$$P(V, NP, PP \mid VP) = \frac{count(VP \rightarrow V\ NP\ PP)}{count(VP \rightarrow \backslash *)}$$

(c) Die aus den Daten extrahierten relativen Häufigkeiten sollen nun zur Erstellung einer probabilistischen kontextfreien Grammatik (PCFG) genutzt werden. Füllen Sie dafür die Lücken in folgender Grammatik mit Ihren berechneten Wahrscheinlichkeiten.

```
1  S → NP VP       [1.0]
2  VP → V NP PP    [{}]
3  VP → V NP       [{}]
4  NP → DET N      [{}]
5  NP → NP PP      [{}]
6  PP → P NP       [1.0]
7
8  DET → "the"       [{}]
9  DET → "a"         [{}]
10 N → "boy"         [0.4]
11 N → "woman"       [0.4]
12 N → "telescope"   [0.2]
13 V → "saw"         [1.0]
14 P → "with"        [1.0]
```

(d) Testen Sie Ihre so erstellte Grammatik nun, indem Sie folgenden Satz parsen:

the boy saw a woman with a telescope

Wenn Sie sich die extrahierten Wahrscheinlichkeiten und das disambiguierte Ergebnis ansehen, überrascht Sie dann das Ergebnis der Syntaxanalyse?

Aufgabe 3 Weiterverarbeitung syntaktischer Analysen

In dieser Aufgabe sollen Sie die Ausgaben eines state-of-the-art-Parsers, nämlich des Stanford Parsers, weiterverarbeiten.

Mit dem Ziel, Sie erst einmal mit den typischen Strukturen einer solchen Aufgabe vertraut zu machen, sollen Sie in dieser Aufgabe lediglich entscheiden, ob die Eingabe einen Infinitivsatz mit Objekt enthält.

Zur Klarheit betrachten Sie die folgenden positiven und negativen Beispiele:

+ Er beabsichtigt , den Kuchen ganz alleine zu essen .
+ Er behauptet , ihn gesehen zu haben .
– Er glaubt , nach Hause zu fliegen .
– Zu fliegen ist schön .
– Er will gehen .

Anmerkung: Orientieren Sie sich bei der Benutzung des Stanford Parsers am Jupyter Notebook dieser Woche.

Hausaufgaben

Aufgabe 4 PCFGs und Viterbi

Beantworten Sie die folgenden grundlegenden Fragen über die heute verwendeten Technologien.

(a) Welche der folgenden Bedingungen wird an eine PCFG gestellt?

- ◯ Die Summe aller Regelwahrscheinlichkeiten für jede LHS ist jeweils 1.
- ◯ Die Summe aller Regelwahrscheinlichkeiten für jede RHS ist jeweils 1.
- ◯ Die Summe aller Regelwahrscheinlichkeiten innerhalb einer Grammatik ist 1.

(b) Was ist die Aufgabe des Viterbi-Algorithmus?

- ◯ Bestimmung der Köpfe und Dependenzrelationen
- ◯ Bestimmung des wahrscheinlichsten Syntaxbaums
- ◯ Finden aller Konstituenten

Aufgabe 5 Fragen zu NLTK-08-extras, 2.9 ("Viterbi-Parser")

(a) Betrachten Sie die folgende im Kapitel gegebene PCFG:

```
1 grammar = nltk.PCFG.fromstring('''
2   NP  -> NNS [0.5] | JJ NNS [0.3] | NP CC NP [0.2]
3   NNS -> "cats" [0.1] | "dogs" [0.2] | "mice" [0.3]
4   NNS -> NNS CC NNS [0.4]
5   JJ  -> "big" [0.4] | "small" [0.6]
6   CC  -> "and" [0.9] | "or" [0.1]
7   ''')
8 viterbi_parser = nltk.ViterbiParser(grammar)
9
```

```
10 sent = 'big cats and dogs'.split()
11 for tree in viterbi_parser.parse(sent):
12     print(tree)
13 # (NP (JJ big) (NNS (NNS cats) (CC and) (NNS dogs))) (p=***)
```

Berechnen Sie die Wahrscheinlichkeit für die Ableitung in der letzten Zeile.

(b) Betrachten Sie das dazugehörige Tracing-Output des ViterbiParsers:

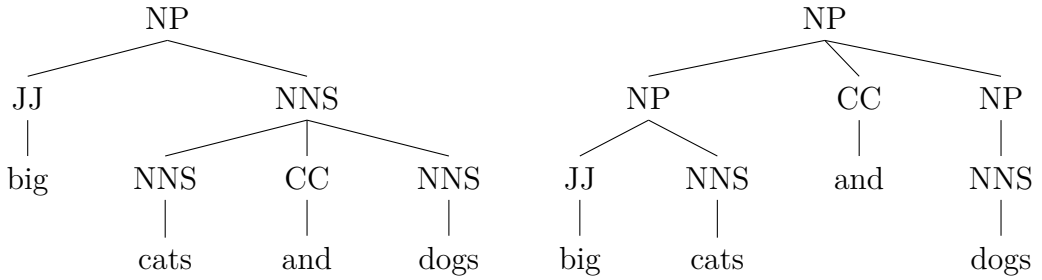
```
>>> sent = 'big cats and dogs'.split()
>>> viterbi_parser = nltk.ViterbiParser(grammar, trace = 3)
>>> for tree in viterbi_parser.parse(sent):
>>>     print(tree)

Inserting tokens into the most likely constituents table...
   Insert: |=...| big
   Insert: |.=..| cats
   Insert: |..=.| and
   Insert: |...=| dogs
Finding the most likely constituents spanning 1 text elements..
   Insert: |=...| JJ -> 'big' [0.4]                0.4000000000
   Insert: |.=..| NNS -> 'cats' [0.1]              0.1000000000
   Insert: |.=..| NP -> NNS [0.5]                  0.0500000000
   Insert: |..=.| CC -> 'and' [0.9]                0.9000000000
   Insert: |...=| NNS -> 'dogs' [0.2]              0.2000000000
   Insert: |...=| NP -> NNS [0.5]                  0.1000000000
Finding the most likely constituents spanning 2 text elements..
   Insert: |==..| NP -> JJ NNS [0.3]               0.0120000000
Finding the most likely constituents spanning 3 text elements..
   Insert: |.===| NP -> NP CC NP [0.2]             0.0009000000
   Insert: |.===| NNS -> NNS CC NNS [0.4]          0.0072000000
   Insert: |.===| NP -> NNS [0.5]                  0.0036000000
  Discard: |.===| NP -> NP CC NP [0.2]             0.0009000000
  Discard: |.===| NP -> NP CC NP [0.2]             0.0009000000
Finding the most likely constituents spanning 4 text elements..
   Insert: |====| NP -> JJ NNS [0.3]               0.0008640000
  Discard: |====| NP -> NP CC NP [0.2]             0.0002160000
  Discard: |====| NP -> NP CC NP [0.2]             0.0002160000
(NP (JJ big) (NNS (NNS cats) (CC and) (NNS dogs))) (p=***)
```

Warum werden die Analysen in den Discard-Zeilen verworfen?

- ◯ Das Verwerfen verhindert eine Endlosrekursion.
- ◯ Die Analysen sind mit der gegebenen Grammatik nicht möglich.
- ◯ Ihre Wahrscheinlichkeiten sind zu gering.
- ◯ Sie analysieren die Eingabesequenz nicht komplett.

(c) Ein statistischer ChartParser findet folgende zwei Ableitungen für die NP *big cats and dogs*:

```
1 (NP (JJ big) (NNS (NNS cats) (CC and) (NNS dogs)))
2 (NP (NP (JJ big) (NNS cats)) (CC and) (NP (NNS dogs)))
```



- Um welchen Parser kann es nicht handeln?
  - ◯ `InsideChartParser`
  - ◯ `InsideChartParser` mit `beam-size`
  - ◯ `LongestChartParser`
  - ◯ `ViterbiParser`
- Nach welchem Kriterium wird beim Parsen mit dem `InsideChartParser` (= Lowest-Cost-First-Strategie) die *edge queue* sortiert?
  - ◯ nach der Länge der Ableitung
  - ◯ nach der Wahrscheinlichkeit
  - ◯ nach der `beam-size`
- Um welche Art der Ambiguität handelt es sich bei den beiden gefundenen Ableitungen?
  - ◯ Koordinationsambiguität
  - ◯ PP-Attachment-Ambiguität
  - ◯ Temporale Ambiguität