Fakultät
Humanwissenschaften

Maschinelle Übersetzung

Alignment - IBM Model 1

Computerlinguistische Techniken

Programmierprojekt

Autor: Le Duyen Sandra Vu
Matr.-Nr.: 768693

Prüfer: Prof. Dr. Alexander Koller

Abgabedatum: 18. März 2016

Inhaltsverzeichnis

1 Einleitung

Word Alignment (bzw. Bitext Word Alignment [7]) ist eines der Hauptaufgaben der maschinellen Übersetzung. Ziel dabei ist es Wörter aus einer Sprache mit deren entsprechenden Übersetzungen zusammenzufassen. Die Methoden zur Bewältigung dieser Aufgabe sind vielfältig. Eine Umsetzungsmöglichkeit bietet hierbei das IBM alignment Model 1, welches das erste und einfachste Model von insgesamt 5 offiziellen IBM Modellen ist [8, 9]. Das IBM Model 1 verwendet im Gegensatz zu den anderen IBM Modellen nur die Lexikalische-Übersetzung, welches einen Nachteil mit sich zieht, wenn beispielsweise Sprichwörter übersetzt werden sollen [9]. Im Vergleich zu anderen Umsetzungen wie zum Beispiel den Aligner, der den Dice's Coefficient nutzt, bietet das IBM Model 1 den Vorteil, dass die Wörter in einem Satz als Übersetzungsmöglichkeit gegeneinander konkurrieren [1].

Wie genau der Algorithmus aussieht und funktioniert, soll im 2. Kapitel dargestellt werden. Darauf folgt im 3. und somit letzen Kapitel die Auseinandersetzung mit der Implementierung des Models in Python.

2 Algorithmus

Ziel des IBM Model 1 ist die Schätzung der Übersetzungswahrscheinlichkeiten $t(e|f)$ aus einem parallelen Korpus. Das Problem hierbei ist jedoch das Fehlen der Alignments für diese Berechnung. Mithilfe des EM-Algorithmus soll diese Problematik behoben werden. Dabei wird zunächst das Modell initialisiert, dann werden Wahrscheinlichkeiten zugeordnet und zuletzt die Parameter aus den Daten geschätzt. Die letzten beiden Schritte, die auch als Expectation-Step und Maximization-Step bekannt sind, werden schließlich optimalerweise so oft wiederholt bis eine Konvergenz erzielt wurde [2].

Im folgenden wird unter Verwendung einer Zusammenfassung von Jan Tore Lønning [6] der Algorithmus genauer erklärt.

1. Schritt: Einheitliche Initialisierung des Models
Alle Parameter des Models t bekommen den selben Wert zugewiesen.

Als Beispiel: $t(f|e) = 1.0$

2. Schritt: Expectation-Step - Erster Durchgang
Die Wahrscheinlichkeitsverteilung für die verschiedenen Alignments werden anhand von t geschätzt (dies ist für den ersten Durchlauf unbedeutend, da zu diesem Zeitpunkt alle Alignments gleichwertig sind).

Als Beispiel: Wenn $t(f_i|e_k) = 3t(f_j|e_k)$, dann sollten die Alignments von f_i zu e_k drei mal wahrscheinlicher sein als die Alignments von f_j zu e_k.

3. Schritt: Maximization-Step
Es wird auf der Basis von t gezählt, wie oft ein Wort e als f übersetzt wurde, dies ist dann der Fractional-Count.

Gegeben sei ein Satzpaar (e_j, f_j), wenn e in e_j und f in f_j vorkommt, dann wird angenommen, dass ein Alignment von f zu e existiert und dieses Alignment a eine Wahrscheinlichkeit P(a) besitzt. Unter Verwendung dieses Alignments wird gezählt, dass e als f "P(a) mal" übersetzt wurde.

Als Beispiel: Wenn P(a) gleich 0.01, dann wird jedes mal 0.01 zum Count, der angibt, wie oft e als f übersetzt wurde, dazu addiert. Nachdem dies für alle Alignments berechnet wurde, wird t neu berechnet.

4. Schritt: Expectation-Step - weitere Durchgänge
Mithilfe der neuen Übersetzungswahrscheinlichkeiten, kann nun für jeden Satz die beste Wahrscheinlichkeitsverteilung aller möglichen Alignments geschätzt werden. Im Gegensatz zum ersten Durchlauf ist es nun auch möglich für jedes Alignment eine Wahrscheinlichkeit anhand von t zu berechnen. Am Ende werden darüber hinaus noch alle Werte Normalisiert, damit sich die Wahrscheinlichkeiten jedes Satzes zu 1 aufsummieren.

5. Schritt: Beliebige Wiederholung der EM-Schritte

a) Schritt 2a wird wiederholt. Auf Basis der Alignment-Wahrscheinlichkeiten, die zuvor im Schritt 2b geschätzt wurden, können nun auch neue Übersetzungswahrscheinlichkeiten t berechnet werden.

b) Aus den neuen Übersetzungswahrscheinlichkeiten t werden darauf folgend neue Alignment-Wahrscheinlichkeiten berechnet.

Werte von 0.9 sind eher eine Seltenheit, dennoch reichen meist 2-3 Iterationen um ein Ergebnis zu erzielen. Desto öfter iteriert wird, desto signifikanter werden die Werte, wodurch auch ein besserer Ergebnis erzielt wird.

3 Implementierung

Die Implementierungsaufgabe ist angelehnt an der "Alignment Challenge Problem 1" Aufgabenstellung aus mt-class.org [1]. Schwerpunkt ist dabei die Implementierung des IBM Model 1 in einer beliebiger Programmiersprache. Dazu wurden des Weiteren 3 Python Programme von Adam Lopez bereitgestellt. Zum einen das Programm "align",

welches eine Implementierung eines Aligners, der den Dice's Coefficient nutz, ist. Zum anderen die Programme "check-alignments" und "score-alignments", die zur Prüfung des eigenen IBM Model 1 Programms dienen. "check-alignments" prüft dabei die Formatierung der Alignments und "score-alignments" berechnet precision, recall und AER (Alignment Error Rate).

Der Datensatz, der zum Trainieren dient, besteht aus insgesamt 100.000 Sätzen (eng/fr) aus dem Canadian Hansards Korpus. Für die Evaluierung stehen ordnungsgemäße Alignments von 37 Sätzen zur Verfügung.

3.1 Umsetzung

Da bereits ein Aligner in Python gegeben worden war, wurde dieser Code für die Implementierung des IBM Model 1 angepasst. Der Code für die Kommandozeilen-Optionen, Daten-Generierung und Print-Funktion blieben dabei weitgehend gleich.

Der eigentliche Umsetzung richtet sich nach 2 Pseudocodes, die den Algorithmus auf verschiedene Arten darstellen, aber im Grunde genommen äquivalent sind.

Im folgenden ist der Pseudocode aus den Vortragsfolien von Philipp Koehn [3] zu sehen:

Input: set of sentence pairs (**e**, **f**)
Output: translation prob. $t(e|f)$

initialize $t(e|f)$ uniformly
while not converged **do**
  // initialize
  count$(e|f) = 0$ **for all** e, f
  total$(f) = 0$ **for all** f
  for all sentence pairs (**e**,**f**) **do**
    // compute normalization
    for all words e in **e** **do**
      s-total$(e) = 0$
      for all words f in **f** **do**
        s-total(e) += $t(e|f)$
      end for
    end for
    // collect counts
    for all words e in **e** **do**
      for all words f in **f** **do**
        count$(e|f)$ += $\frac{t(e|f)}{\text{s-total}(e)}$
        total(f) += $\frac{t(e|f)}{\text{s-total}(e)}$
      end for
    end for
  end for
  // estimate probabilities
  for all foreign words f **do**
    for all English words e **do**
      $t(e|f) = \frac{\text{count}(e|f)}{\text{total}(f)}$
    end for
  end for
end while

Etwas kompakter wirkt dahingegen der Pseudocode aus den Vortragsfolien von Chris Callison-Burch [2]:

```
initialize t(e|f) uniformly
do
  set count(e|f) to 0 for all e,f
  set total(f) to 0 for all f
  for all sentence pairs (e_s,f_s)
    for all words e in e_s
      total_s = 0
      for all words f in f_s
        total_s += t(e|f)
    for all words e in e_s
      for all words f in f_s
        count(e|f) += t(e|f) / total_s
        total(f)   += t(e|f) / total_s
  for all f in domain( total(.) )
    for all e in domain( count(.|f) )
      t(e|f) = count(e|f) / total(f)
until convergence
```

Die Kommentierung in der Model1.py Datei zeigt genauestens an, wie die einzelnen Zeilen des Pseudocodes umgesetzt wurden. Zudem muss erwähnt werden, dass die Iterationsanzahl nicht von der Konvergenz abhängig programmiert worden ist. Dies hängt damit zusammen, dass kein fester Konvergenz-Wert existiert. Die Iterationsanzahl und der Schwellenwert können dafür vom Nutzer angegeben werden und beeinflussen je nachdem das Ergebnis.

3.2 Ausführung

Für das Ausführen der Model1.py muss zuerst das Terminal gestartet werden. Danach muss in den Pfad dirigiert werden in dem die Datei liegt.

Der Befehl zum Starten des Programms lautet: python Model1.py
Zusätzlich können noch Kommandozeilen-Optionen zur Anpassung von Variablen angegeben werden.

-d	–data	default= ”data/hansards”	Prefix des Daten-Dateinamens
-e	–english	default= ”e”	Suffix der Englisch-Datei
-f	–french	default= ”f”	Suffix der Französisch-Datei
-t	–threshold	default= 0.2	Schwellenwert
-l	–loop	default= 3	Anzahl der Iterationen
-n	–num_sentences	default=1000	Anzahl der Sätze

Als Beispiel: python Model1.py -l 5 -t 0.5

Die berechneten Alignments werden dann im Terminal ausgegeben. Im Fall einer notwendigen Weiterverarbeitung müssen die Alignments in eine Datei abgespeichert werden, außer die Programme "check-alignments" und "score-alignments" werden direkt in Folge ausgeführt.

Als Beispiel: python Model1.py -l 5 -t 0.5 > out.txt
und darauf folgend python score-alignments < out.

bzw.: python Model1.py -l 5 -t 0.5 — python score-alignments

3.3 Auswertung

Tabelle 1: IBM Model 1

Satzanzahl	Iterationen	Schwellwert	Precision	Recall	AER
1000	1	0.1	0.165	0.020	0.922
1000	1	0.5	Keine	Alignments	gefunden
1000	3	0.1	0.235	0.698	0.679
1000	3	0.5	0.888	0.038	0.918
1000	10	0.1	0.286	0.798	0.626
1000	10	0.5	0.675	0.511	0.411

Bei einem hohen Schwellwert kann es dazu kommen, dass nur wenige oder sogar gar keine Alignments gefunden werden, daraus folgt ein niedriger Recall. Die Alignments, die jedoch mit dem hohen Schwellwert gefunden wurden, ziehen dafür eine hohe Precision mit sich. Die niedrigste hier erzielte Alignment Error Rate (AER) beträgt 0.411 und ist deutlich unter dem niedrigsten AER Wert 0.680 des Dice's coefficient-Aligners.

Tabelle 2: Dice's coefficient

Satzanzahl	Schwellwert	Precision	Recall	AER
1000	0.1	0.155	0.928	0.803
1000	0.5	0.247	0.653	0.680
1000	0.9	0.365	0.224	0.704

3.4 Probleme

Da die komplette Matrix mittels Default-Dictionaries generiert wird, ist die Berechnung mit über 5.000 Sätzen bereits zeitaufwändig und Speicherplatz raubend. Zieht man die zwei Quellen in betracht [4, 5] ist es nicht möglich eine Matrix mit der Größe 50.000 x 50.000 oder größer auf einmal zu generieren. Dies bedeutet, dass die Daten entweder Zwischengespeichert werden müssen, dass ein effizienterer Container genutzt werden muss oder dass die Größe der Matrix bestmöglich reduziert werden muss.

4 Quellen

[1] Aufgabenstellung: Alignment Challenge Problem 1, 25.06.2015.
http://mt-class.org/jhu-2014/hw1.html

[2] Folien: Chris Callison-Burch, Philipp Koehn: Machine translation: Word-based models and the EM algorithm, 03.12.2007.
https://www.cs.jhu.edu/∼jason/465/PowerPoint/lect32a-mt-word-based-models.pdf

[3] Folien: Philipp Koehn: IBM Model 1 and the EM Algorithm, 09.02.2016.
http://mt-class.org/jhu/slides/lecture-ibm-model1.pdf

[4] Forum: Large Two Dimensional Array, 28.01.2014.
http://www.gossamer-threads.com/lists/python/python/1115759

[5] Forum: Is it possible to create a 1million x 1 million matrix using numpy? [duplicate], 14.06.2011.
http://stackoverflow.com/questions/6338986/is-it-possible-to-create-a-1million-x-1-million-matrix-using-numpy

[6] Notiz: Jan Tore Lønning, Note on EM-training of IBM-model 1, Herbst 2012.
http://www.uio.no/studier/emner/matnat/ifi/INF5820/h12/undervisningsmateriale/emtraining.pdf

[7] Wikipedia: Bitext word alignment.
https://en.wikipedia.org/wiki/Bitext_word_alignment

[8] Wikipedia: IBM alignment models.
https://en.wikipedia.org/wiki/IBM_alignment_models

[9] Youtube: Computational Linguistics I: Machine Translation (IBM Model 1), 21.10.2013 - Hochgeladen von Jordan Boyd-Graber.
https://www.youtube.com/watch?v=nKqyU0bUKTQ

[10] Youtube: 2. IBM Model 1 - Part I,
06.05.2013 - Hochgeladen von Arnaldo Pedro Figueira Figueira.
https://www.youtube.com/watch?v=PEvRedAiF-E

[11] Youtube: 3. IBM Model 1 - Part II,
06.05.2013 - Hochgeladen von Arnaldo Pedro Figueira Figueira.
https://www.youtube.com/watch?v=n58-akQIlQ4