

Möglichkeiten der automatischen Sprachverarbeitung mit Django

Julian Moritz, jumo@gmx.de

März 2009 / Leipzig / Python Stammtisch

Inhalt

- 1 Einführung
- 2 Verarbeitung
- 3 Django
- 4 Weiterführendes

Einführung - ASV

Was macht man bei der automatischen Sprachverarbeitung?

- **Speech2Text, Text2Speech**
- (automatische) Rechtschreibkorrektur
- Übersetzung, Erstellung von Wörterbüchern
- Aufbereitung von großen Textmengen, Durchsuchen, Vergleichen von Dokumenten

Einführung - ASV

Was macht man bei der automatischen Sprachverarbeitung?

- Speech2Text, Text2Speech
- (automatische) Rechtschreibkorrektur
- Übersetzung, Erstellung von Wörterbüchern
- Aufbereitung von großen Textmengen, Durchsuchen, Vergleichen von Dokumenten

Einführung - ASV

Was macht man bei der automatischen Sprachverarbeitung?

- Speech2Text, Text2Speech
- (automatische) Rechtschreibkorrektur
- Übersetzung, Erstellung von Wörterbüchern
- Aufbereitung von großen Textmengen, Durchsuchen, Vergleichen von Dokumenten

Einführung - ASV

Was macht man bei der automatischen Sprachverarbeitung?

- Speech2Text, Text2Speech
- (automatische) Rechtschreibkorrektur
- Übersetzung, Erstellung von Wörterbüchern
- Aufbereitung von großen Textmengen, Durchsuchen, Vergleichen von Dokumenten

Einsatz bei der Webprogrammierung

Ähnliche Themen für: Baudarlehnen von der Deutschen Bank!

Thema	Autor	
Meine Bank will mich zum Onlinebanking überreden!	hausmann	Sonstiges
Entstehen bei der Bank Gebühren, wenn das Haus verkauft wird?	Sven H.	Baufinanz

MEHR ÜBER...

**Lufthansa Flugbegleiter
Unabhängige
Flugbegleiter
Organisation**

zu SPIEGEL WISSEN ▶▶

Allerdings soll Lufthansa künftig Flugpläne so wenig wie möglich verändern. Nur so hätten Mitarbeiter die Möglichkeit, ihr Privatleben zu planen, hatte Müller nach den gescheiterten Verhandlungen gesagt.

Das Unternehmen hatte dem

sen/wissenschaft/astronomie/tid-13567/kepler-mission-auf-der-suche-nach-der-zweiten-er

Zerlegung

- Zunächst muss ein Text zerlegt werden, optional in Sätze, auf jeden Fall in Wörter.
- Satzzerlegung: Nach Satzende-Zeichen abschneiden, aber Abkürzungen beachten!
- Wortzerlegung: Schwierig, aber machbar. Mein Ansatz: Menge von Nicht-Wortzeichen definieren, und Teilmenge davon mit Zeichen, die innerhalb von Wörtern vorkommen dürfen.
(Nichtwortzeichen sind zum Beispiel , . ? ! - + \$, aber die Zeichen . - ' dürfen innerhalb von Wörtern vorkommen, zum Beispiel bei *F.D.P*, *don't*, *H-Milch*.)

Zerlegung

- Zunächst muss ein Text zerlegt werden, optional in Sätze, auf jeden Fall in Wörter.
- Satzzerlegung: Nach Satzende-Zeichen abschneiden, aber Abkürzungen beachten!
- Wortzerlegung: Schwierig, aber machbar. Mein Ansatz: Menge von Nicht-Wortzeichen definieren, und Teilmenge davon mit Zeichen, die innerhalb von Wörtern vorkommen dürfen.
(Nichtwortzeichen sind zum Beispiel , . ? ! - + \$, aber die Zeichen . - ' dürfen innerhalb von Wörtern vorkommen, zum Beispiel bei *F.D.P*, *don't*, *H-Milch*.)

Zerlegung

- Zunächst muss ein Text zerlegt werden, optional in Sätze, auf jeden Fall in Wörter.
- Satzzerlegung: Nach Satzende-Zeichen abschneiden, aber Abkürzungen beachten!
- Wortzerlegung: Schwierig, aber machbar. Mein Ansatz: Menge von Nicht-Wortzeichen definieren, und Teilmenge davon mit Zeichen, die innerhalb von Wörtern vorkommen dürfen.
(Nichtwortzeichen sind zum Beispiel , . ? ! - + \$, aber die Zeichen . - ' dürfen innerhalb von Wörtern vorkommen, zum Beispiel bei *F.D.P*, *don't*, *H-Milch*.)

Invertierte Wortliste

- Die Speicherung erfolgt in einer invertierten Wortliste. Es werden also nicht mehr die Dokumente mit ihren Wörtern gespeichert, sondern die Wörter und deren Dokumente.

Wort	Dokumente
1	1 (2), 2 (4), 3 (2)
2	1 (5), 3 (1)
3	3 (3)

- Weitere Informationen für ein Dokument können gespeichert werden (Häufigkeit, Positionen)

Invertierte Wortliste

- Die Speicherung erfolgt in einer invertierten Wortliste. Es werden also nicht mehr die Dokumente mit ihren Wörtern gespeichert, sondern die Wörter und deren Dokumente.

Wort	Dokumente
1	1 (2), 2 (4), 3 (2)
2	1 (5), 3 (1)
3	3 (3)

- Weitere Informationen für ein Dokument können gespeichert werden (Häufigkeit, Positionen)

Invertierte Wortliste

- Die Speicherung erfolgt in einer invertierten Wortliste. Es werden also nicht mehr die Dokumente mit ihren Wörtern gespeichert, sondern die Wörter und deren Dokumente.

Wort	Dokumente
1	1 (2), 2 (4), 3 (2)
2	1 (5), 3 (1)
3	3 (3)

- Weitere Informationen für ein Dokument können gespeichert werden (Häufigkeit, Positionen)

Signifikante Terme für ein Dokument I

- Ein Maß ist TFIDF. Grundidee: Je häufiger ein Wort in einem Dokument auftaucht und je seltener in dem Gesamtkorpus, desto Signifikanter ist es.
- Erster Wert ist tf , hierzu wird die relative Häufigkeit normalisiert: Wir betrachten Dokument 3



Wort	normalisiert nach Summe	normalisiert nach Maximum
1	$tf_{1,\Sigma} = \frac{2}{6} = \frac{1}{3}$	$tf_{1,\max} = \frac{2}{3}$
2	$tf_{2,\Sigma} = \frac{1}{6}$	$tf_{2,\max} = \frac{1}{3}$
3	$tf_{3,\Sigma} = \frac{3}{6} = \frac{1}{2}$	$tf_{3,\max} = \frac{3}{3} = 1$

Signifikante Terme für ein Dokument I

- Ein Maß ist TFIDF. Grundidee: Je häufiger ein Wort in einem Dokument auftaucht und je seltener in dem Gesamtkorpus, desto Signifikanter ist es.
- Erster Wert ist tf , hierzu wird die relative Häufigkeit normalisiert: Wir betrachten Dokument 3



Wort	normalisiert nach Summe	normalisiert nach Maximum
1	$tf_{1,\Sigma} = \frac{2}{6} = \frac{1}{3}$	$tf_{1,\max} = \frac{2}{3}$
2	$tf_{2,\Sigma} = \frac{1}{6}$	$tf_{2,\max} = \frac{1}{3}$
3	$tf_{3,\Sigma} = \frac{3}{6} = \frac{1}{2}$	$tf_{3,\max} = \frac{3}{3} = 1$

Signifikante Terme für ein Dokument I

- Ein Maß ist TFIDF. Grundidee: Je häufiger ein Wort in einem Dokument auftaucht und je seltener in dem Gesamtkorpus, desto Signifikanter ist es.
- Erster Wert ist tf , hierzu wird die relative Häufigkeit normalisiert:
Wir betrachten Dokument 3



Wort	normalisiert nach Summe	normalisiert nach Maximum
1	$tf_{1,\Sigma} = \frac{2}{6} = \frac{1}{3}$	$tf_{1,\max} = \frac{2}{3}$
2	$tf_{2,\Sigma} = \frac{1}{6}$	$tf_{2,\max} = \frac{1}{3}$
3	$tf_{3,\Sigma} = \frac{3}{6} = \frac{1}{2}$	$tf_{3,\max} = \frac{3}{3} = 1$

Signifikante Terme für ein Dokument II

- Zweiter Wert ist *idf*, die inverse Dokumentenfrequenz.
Allgemeine Formel:

$$idf = \log\left(\frac{|d|}{|d : w \in d|}\right)$$

mit d = Anzahl der Dokumente.

Wort	Inverse Dokumentenfrequenz	$tf_{\Sigma} * idf$	$tf_{\max} * idf$
1	$idf_1 = \log\left(\frac{3}{3}\right) = \log(1)$	0	0
2	$idf_2 = \log\left(\frac{3}{2}\right) = \log(1.5)$	0.03	0.06
3	$idf_3 = \log\left(\frac{3}{1}\right) = \log(3)$	0.24	0.48

Signifikante Terme für ein Dokument II

- Zweiter Wert ist *idf*, die inverse Dokumentenfrequenz.
Allgemeine Formel:

$$idf = \log\left(\frac{|d|}{|d : w \in d|}\right)$$

mit d = Anzahl der Dokumente.

Wort	Inverse Dokumentenfrequenz	$tf_{\Sigma} * idf$	$tf_{\max} * idf$
1	$idf_1 = \log\left(\frac{3}{3}\right) = \log(1)$	0	0
2	$idf_2 = \log\left(\frac{3}{2}\right) = \log(1.5)$	0.03	0.06
3	$idf_3 = \log\left(\frac{3}{1}\right) = \log(3)$	0.24	0.48

Signifikante Terme für ein Dokument III

- Die Nullhypothese besagt, dass ein Wort in einem Dokument und einem Referenzkorpus gleich wahrscheinlich auftreten.
- Mit der Log-Likelihood-Methode wird die Überraschung gemessen, wie häufig das Wort tatsächlich in dem Dokument auftritt.
- Formel:

$$\text{sig}(w) = \frac{k(\log *k - \log \lambda - 1)}{\log n}$$

mit k = Häufigkeit von Wörtern in Dokument, n = Länge des Textes, p = relative Häufigkeit von Wort in Referenzkorpus und $\lambda = n * p$.

Signifikante Terme für ein Dokument III

- Die Nullhypothese besagt, dass ein Wort in einem Dokument und einem Referenzkorpus gleich wahrscheinlich auftreten.
- Mit der Log-Likelihood-Methode wird die Überraschung gemessen, wie häufig das Wort tatsächlich in dem Dokument auftritt.
- Formel:

$$\text{sig}(w) = \frac{k(\log *k - \log \lambda - 1)}{\log n}$$

mit k = Häufigkeit von Wörtern in Dokument, n = Länge des Textes, p = relative Häufigkeit von Wort in Referenzkorpus und $\lambda = n * p$.

Signifikante Terme für ein Dokument III

- Die Nullhypothese besagt, dass ein Wort in einem Dokument und einem Referenzkorpus gleich wahrscheinlich auftreten.
- Mit der Log-Likelihood-Methode wird die Überraschung gemessen, wie häufig das Wort tatsächlich in dem Dokument auftritt.
- Formel:

$$\text{sig}(w) = \frac{k(\log *k - \log \lambda - 1)}{\log n}$$

mit k = Häufigkeit von Wörtern in Dokument, n = Länge des Textes, p = relative Häufigkeit von Wort in Referenzkorpus und $\lambda = n * p$.

Implementierung in Django I

Zunächst das Document-Model:

```
1 class Document(models.Model):
2     content_type = models.ForeignKey(ContentType, verbose_name=_("content type"))
3     object_id = models.PositiveIntegerField(_("object id"))
4     content_object = generic.GenericForeignKey("content_type", "object_id")
5     timestamp_created = models.DateTimeField(_("timestamp created"), auto_now_add=True, editable=False)
6
7     def __unicode__(self):
8         return unicode(self.content_object)
9
10    def get_absolute_url(self):
11        return self.content_object.get_absolute_url()
```

Desweiteren das Word-Model:

```
1 class Word(models.Model):
2     word = models.CharField(_("word"), max_length=255, unique=True)
3     documents = models.ManyToManyField(Document, verbose_name=_("documents"), through=Type)
```

Und nun noch der Type:

```
1 class Type(models.Model):
2     document = models.ForeignKey(Document, verbose_name=_("document"))
3     word = models.ForeignKey("Word", verbose_name=_("word"))
4     count = models.PositiveIntegerField(_("count"))
5     weight = models.FloatField(_("weight"))
```

Implementierung in Django I

Zunächst das Document-Model:

```
1 class Document(models.Model):
2     content_type = models.ForeignKey(ContentType, verbose_name=_("content type"))
3     object_id = models.PositiveIntegerField(_("object id"))
4     content_object = generic.GenericForeignKey("content_type", "object_id")
5     timestamp_created = models.DateTimeField(_("timestamp created"), auto_now_add=True, editable=False)
6
7     def __unicode__(self):
8         return unicode(self.content_object)
9
10    def get_absolute_url(self):
11        return self.content_object.get_absolute_url()
```

Desweiteren das Word-Model:

```
1 class Word(models.Model):
2     word = models.CharField(_("word"), max_length=255, unique=True)
3     documents = models.ManyToManyField(Document, verbose_name=_("documents"), through=Type)
```

Und nun noch der Type:

```
1 class Type(models.Model):
2     document = models.ForeignKey(Document, verbose_name=_("document"))
3     word = models.ForeignKey("Word", verbose_name=_("word"))
4     count = models.PositiveIntegerField(_("count"))
5     weight = models.FloatField(_("weight"))
```

Implementierung in Django I

Zunächst das Document-Model:

```
1 class Document(models.Model):
2     content_type = models.ForeignKey(ContentType, verbose_name=_("content type"))
3     object_id = models.PositiveIntegerField(_("object id"))
4     content_object = generic.GenericForeignKey("content_type", "object_id")
5     timestamp_created = models.DateTimeField(_("timestamp created"), auto_now_add=True, editable=False)
6
7     def __unicode__(self):
8         return unicode(self.content_object)
9
10    def get_absolute_url(self):
11        return self.content_object.get_absolute_url()
```

Desweiteren das Word-Model:

```
1 class Word(models.Model):
2     word = models.CharField(_("word"), max_length=255, unique=True)
3     documents = models.ManyToManyField(Document, verbose_name=_("documents"), through=Type)
```

Und nun noch der Type:

```
1 class Type(models.Model):
2     document = models.ForeignKey(Document, verbose_name=_("document"))
3     word = models.ForeignKey("Word", verbose_name=_("word"))
4     count = models.PositiveIntegerField(_("count"))
5     weight = models.FloatField(_("weight"))
```

Implementierung in Django II

Das Speichern funktioniert via Signal. Signals sind so etwas ähnliches wie SQL-Trigger; man kann dafür sorgen, dass eine bestimmte Funktion beim Speichern einer (beliebigen) Model-Instanz aufgerufen wird.

```
1 def textentry_save_handler(sender, **kwargs):
2     textentry = kwargs["instance"]
3     text = "%s\r%s" % (textentry.content, textentry.headline)
4     tp = Textprocessing()
5     tp.process(text, textentry)
6
7 def delete_handler(sender, **kwargs):
8     instance = kwargs["instance"]
9     tp = Textprocessing()
10    tp.delete_words(instance)
11    tp.delete_document(instance)
12 #later in the code:
13 from django.db.models.signals import post_save, post_delete
14 post_save.connect(textentry_save_handler, sender=TextEntry)
15 post_delete.connect(delete_handler, sender=TextEntry)
```

Implementierung in Django III

Das Anzeigen wird via Template-Tag gelöst

```
1 {% get_sigterms for entry.textentry as sigtermelist %}
2 Tags:
3 {% for term in sigtermelist|slice:"7" %}
4   {{term.word}}{% if not forloop.last %} | {% endif %}
5 {% endfor %}
```

```
1 {% get_sigterms as sigtermelist %}
2 {% for term in sigtermelist|slice:"10"|list|shuffle %}
3   <span style="font-size:{{ term.weight|multiply:"30"|integer}}px">{{ term.word}}</span> {% endfor %}
```

```
1 #sigtermelist for a very document:
```

```
2 qs = Word.objects.filter(documents=doc).filter(type__weight__gt=0.0, type__document=doc).order_by("-type__weight")
```

```
3 #sigtermelist in general:
```

```
4 qs = Word.objects.annotate(weight=Avg("type__weight")).filter(weight__gt=0.0).order_by("-weight")
```

Dokumentenähnlichkeiten I

Hierzu wird ein Dokument als Vektor betrachtet. Gibt es zwei Dokumente (d_1, d_2) und insgesamt vier Wörter (w_1, w_2, w_3, w_4) mit folgenden Verteilungen:

$w_1(0.1), w_2(0.3), w_3(0.2) \in d_1, w_1(0.4), w_2(0.2), w_4(0.1) \in d_2$
so sieht der Vektor für d_1 bzw. d_2 wie folgt aus:

$$\vec{d}_1 = (0.1, 0.3, 0.2, 0.0), \vec{d}_2 = (0.4, 0.2, 0.0, 0.1)$$

normiert, so dass $|\vec{v}| = 1$, so:

$$\vec{d}_1 = \left(\sqrt{\frac{1}{6}}, \sqrt{\frac{1}{2}}, \sqrt{\frac{1}{3}}, 0\right), \vec{d}_2 = \left(\sqrt{\frac{4}{7}}, \sqrt{\frac{2}{7}}, 0, \sqrt{\frac{1}{7}}\right)$$

Dokumentenähnlichkeiten II

- Euklidische Distanz:

$$\text{dist}_{\text{Eukl}}(\vec{d}_1, \vec{d}_2) = \sqrt{\sum_{k=1}^n ((w_{k,i} - w_{k,j})^2)}$$

- Skalarprodukt:

$$\text{sim}_{\text{Skal}}(\vec{d}_1, \vec{d}_2) = \sum_{k=1}^n (w_{k,i} * w_{k,j})$$

- Cosinus-Maß:

$$\text{sim}_{\text{Cos}}(\vec{d}_1, \vec{d}_2) = \frac{\sum_{k=1}^n (w_{k,i} * w_{k,j})}{\sqrt{\sum_{k=1}^n (w_{k,i})^2} * \sqrt{\sum_{k=1}^n (w_{k,j})^2}}$$

Wichtig ist bei allen Ähnlichkeitsmaßen mit Wortgewichten, dass die Worthäufigkeiten dem Zipf'schen Gesetz folgen. Für Wörter sortiert nach Häufigkeit gilt: $r * n = k$, also *Rang mal Häufigkeit ist konstant*

Dokumentenähnlichkeiten II

- Euklidische Distanz:

$$\text{dist}_{\text{Eukl}}(\vec{d}_1, \vec{d}_2) = \sqrt{\sum_{k=1}^n ((w_{k,i} - w_{k,j})^2)}$$

- Skalarprodukt:

$$\text{sim}_{\text{Skal}}(\vec{d}_1, \vec{d}_2) = \sum_{k=1}^n (w_{k,i} * w_{k,j})$$

- Cosinus-Maß:

$$\text{sim}_{\text{Cos}}(\vec{d}_1, \vec{d}_2) = \frac{\sum_{k=1}^n (w_{k,i} * w_{k,j})}{\sqrt{\sum_{k=1}^n (w_{k,i})^2} * \sqrt{\sum_{k=1}^n (w_{k,j})^2}}$$

Wichtig ist bei allen Ähnlichkeitsmaßen mit Wortgewichten, dass die Worthäufigkeiten dem Zipf'schen Gesetz folgen. Für Wörter sortiert nach Häufigkeit gilt: $r * n = k$, also *Rang mal Häufigkeit ist konstant*

Dokumentenähnlichkeiten II

- Euklidische Distanz:

$$\text{dist}_{\text{Eukl}}(\vec{d}_1, \vec{d}_2) = \sqrt{\sum_{k=1}^n ((w_{k,i} - w_{k,j})^2)}$$

- Skalarprodukt:

$$\text{sim}_{\text{Skal}}(\vec{d}_1, \vec{d}_2) = \sum_{k=1}^n (w_{k,i} * w_{k,j})$$

- Cosinus-Maß:

$$\text{sim}_{\text{Cos}}(\vec{d}_1, \vec{d}_2) = \frac{\sum_{k=1}^n (w_{k,i} * w_{k,j})}{\sqrt{\sum_{k=1}^n (w_{k,i})^2} * \sqrt{\sum_{k=1}^n (w_{k,j})^2}}$$

Wichtig ist bei allen Ähnlichkeitsmaßen mit Wortgewichten, dass die Worthäufigkeiten dem Zipf'schen Gesetz folgen. Für Wörter sortiert nach Häufigkeit gilt: $r * n = k$, also *Rang mal Häufigkeit ist konstant*

Dokumentenähnlichkeiten II

- Euklidische Distanz:

$$\text{dist}_{\text{Eukl}}(\vec{d}_1, \vec{d}_2) = \sqrt{\sum_{k=1}^n ((w_{k,i} - w_{k,j})^2)}$$

- Skalarprodukt:

$$\text{sim}_{\text{Skal}}(\vec{d}_1, \vec{d}_2) = \sum_{k=1}^n (w_{k,i} * w_{k,j})$$

- Cosinus-Maß:

$$\text{sim}_{\text{Cos}}(\vec{d}_1, \vec{d}_2) = \frac{\sum_{k=1}^n (w_{k,i} * w_{k,j})}{\sqrt{\sum_{k=1}^n (w_{k,i})^2} * \sqrt{\sum_{k=1}^n (w_{k,j})^2}}$$

Wichtig ist bei allen Ähnlichkeitsmaßen mit Wortgewichten, dass die Worthäufigkeiten dem Zipf'schen Gesetz folgen. Für Wörter sortiert nach Häufigkeit gilt: $r * n = k$, also *Bang mal Häufigkeit ist konstant.*

Was ist noch zu tun?

- Zur manuellen Vervollständigung von einer Tag-Liste eventuell ein Widget, das mittels Ajax die automatisch berechneten Tags als Liste anzeigt und so eine Korrektur erlaubt.
- Ähnliches für das Slug-Feld.
- Berechnung ähnlicher Dokumente, Terme (zur Laufzeit? Vorher? Zyklisch? Welche Maße?).
- Implementierung einer Suche mit Query-Analyse.
- Grundformreduktion für die Gewichtsberechnung, Named Entity Recognition für die Worterkennung.

Was ist noch zu tun?

- Zur manuellen Vervollständigung von einer Tag-Liste eventuell ein Widget, das mittels Ajax die automatisch berechneten Tags als Liste anzeigt und so eine Korrektur erlaubt.
- Ähnliches für das Slug-Feld.
- Berechnung ähnlicher Dokumente, Terme (zur Laufzeit? Vorher? Zyklisch? Welche Maße?).
- Implementierung einer Suche mit Query-Analyse.
- Grundformreduktion für die Gewichtsberechnung, Named Entity Recognition für die Worterkennung.

Was ist noch zu tun?

- Zur manuellen Vervollständigung von einer Tag-Liste eventuell ein Widget, das mittels Ajax die automatisch berechneten Tags als Liste anzeigt und so eine Korrektur erlaubt.
- Ähnliches für das Slug-Feld.
- Berechnung ähnlicher Dokumente, Terme (zur Laufzeit? Vorher? Zyklisch? Welche Maße?).
- Implementierung einer Suche mit Query-Analyse.
- Grundformreduktion für die Gewichtsberechnung, Named Entity Recognition für die Worterkennung.

Was ist noch zu tun?

- Zur manuellen Vervollständigung von einer Tag-Liste eventuell ein Widget, das mittels Ajax die automatisch berechneten Tags als Liste anzeigt und so eine Korrektur erlaubt.
- Ähnliches für das Slug-Feld.
- Berechnung ähnlicher Dokumente, Terme (zur Laufzeit? Vorher? Zyklisch? Welche Maße?).
- Implementierung einer Suche mit Query-Analyse.
- Grundformreduktion für die Gewichtsberechnung, Named Entity Recognition für die Worterkennung.

Was ist noch zu tun?

- Zur manuellen Vervollständigung von einer Tag-Liste eventuell ein Widget, das mittels Ajax die automatisch berechneten Tags als Liste anzeigt und so eine Korrektur erlaubt.
- Ähnliches für das Slug-Feld.
- Berechnung ähnlicher Dokumente, Terme (zur Laufzeit? Vorher? Zyklisch? Welche Maße?).
- Implementierung einer Suche mit Query-Analyse.
- Grundformreduktion für die Gewichtsberechnung, Named Entity Recognition für die Worterkennung.

Quellen

- Text Mining: Wissensrohstoff Text. Gerhard Heyer, Uwe Quasthoff, Thomas Wittig. 2006, W3L GmbH.
- <http://www.github.com/feuervogel/djangoproject>

Ende

Fragen?