Information Extraction in der Koch-Domäne

- Regular Expression-based
- Conditional Random Field-based
- Dictionary- and rule-based

Folgend werden Algorithmen zum Extrahieren von Zutaten aus semi-strukturiertem Text vorgestellt, die wir im Kontext der Koch-Domäne gefunden haben. Sind die Zutaten erst einmal von einem Programm extrahiert, kann das Programm diese leicht mit cueML (cueML.html) auszeichnen und zurückschreiben. Das automatische Auszeichnen ist wünschenswert, **da das manuelle Auszeichnen...**

- **zeitaufwendig ist.** Die Erfahrung hat gezeigt, dass ich selber im Schnitt ca. 5 Minuten pro Rezept zum Auszeichnen brauche. Eine SHK, welche nicht im Umgang mit XML geübt war, hat freundlicherweise ebenfalls ein paar Rezepte ausgezeichnet. Sie hat im Schnitt sogar 15 Minuten für ein Rezept gebraucht.
- **fehleranfällig ist.** Diese Feststellung bei mir selbst, wie auch bei der SHK werden u.A. durch (Erdmann et al., 2001) (Quellen.html#manualTaggingErrorProne) und (Greene, 2015) (Quellen.html#CRFNYT) bestätigt.
- **Domänen-spezifisches Wissen benötigt.** Wir sind z. B. über die Zutat *Scorzonerwurzel* gestolpert. Das Nachfragen bei einer Ernährungswissenschaftlerin der Kieler Uni ergab, dass damit wahrscheinlich Scorzone, bzw. Sommer-Trüffel gemeint ist. Später ist uns jedoch aufgefallen, dass Frau Davidis in viele Suppen Scorzonerwurzeln hineingibt. Dies hat bei uns die Frage aufgeworfen, ob sie wirklich oft empfiehlt, teure Sommer-Trüffel in Suppen zu zerkochen. Nach einer weiteren Recherche glauben wir nun, dass sie mit Scorzonerwurzel Scorzonera hispanica bzw. Schwarzwurzel meint, welche sicherlich in einer Suppe angebrachter sind als Sommer-Trüffel.

Regular Expression-based

In (Skip The Pizza auf WordPress.org, 2012) (Quellen.html#SkipThePizza) kombiniert der Autor seine zwei Hobbies Programmieren und Kochen. Um Fragen beantworten zu können, wie *aus wie vielen Zutaten besteht ein Rezept im Schnitt?* oder *welche sind die meist benutzen Zutaten?*, will er die Zutaten aus über 29.000 Rezepten von Recipes wikia (http://recipes.wikia.com/wiki/Recipes_Wiki) extrahieren.

Abb. 1 zeigt, wie die Rezepte in Recipes wikia abgespeichert sind. Offenbar haben die Rezepte die Semi-Struktur, dass jede Zutat innerhalb *[[...]]* eingeschlossen ist. Der Autor nutzt diese Semi-Struktur aus, um mittels Regulärer Ausdrücke die Zutaten zu extrahieren.

Abb. 1: Interne Struktur der Rezepte von Recipes wikia (http://recipes.wikia.com/wiki/Recipes_Wiki)

```
* Makes 6 to 8 servings

== Ingredients ==
* 2 tbsp extra virgin [[olive oil]]
* 3 cloves [[garlic]], finely chopped
* 1½ pounds fresh ripe [[tomato]]es, seeded and chopped (about 3 cups)
* 1 tbsp [[tomato paste]]
* 1 tsp dried [[oregano]]
* ⅛ tsp [[ground red pepper]]
* ½ cup pitted brine cured [[black olives]] coarsely chopped
* 2 tbsp [[capers]]
* [[salt]] and [[pepper]]
* 1 pkg. thin [[spaghetti]] (16 oz)
* grated [[Parmesan cheese]]

== Directions ==
[...]
```

Conditional Random Field-based

Die New York Times (NYT) betreibt eine eigene Kochseite (https://cooking.nytimes.com/). Greene behauptet, dass sie mittels Linear-chain Conditional Random Fields (CRF) in der Lage sind *„automatisch den unstrukturierten Text von Rezepten in strukturierte Daten umzuwandeln“* (Greene, 2015) (Quellen.html#CRFNYT). Dies beinhaltet insbesondere das Extrahieren der Zutaten mit ihren Mengenangaben und Einheiten. Da diese Behauptung genau unserer Problemstellung entspricht, stellen wir zunächst die Idee von CRF vor und gehen anschließend auf die Implementierung von Greene ein.

CRF erklärt

CRF *orakelt* zu einem Vektor von Wörtern ein Vektor von Labels. Im Folgenden bezeichnen wir daher CRF auch als ein Orakel. Wenn wir dem Orakel z. B. den Vektor *[Man, süße, mit, 2, EL, Zucker, .]* geben, orakelt es uns idealerweise die Labels *[OTHER, OTHER, OTHER, QUANTITY, UNIT, INGREDIENT, OTHER]*. Wenn wir so ein Orakel hätten, würden wir unsere Rezepte in solche Wort-Vektoren aufspalten und dem Orakel übergeben. Aus der Antwort des Orakels könnten wir dann sofort die Zutaten mit Mengenangaben und Einheiten ablesen.

Jetzt stellt sich nur noch die Frage, woher man so ein Orakel bekommt. Folgend erläutern wir die Idee, wie so ein Orakel antrainiert werden kann. Eine detaillierte Einführung ist in (Sutton, McCallum, 2012) (Quellen.html#CRFIntroduction) zu finden. Wir fangen damit an, dass wir dem Orakel eine Reihe von Wort-Vektoren geben und ihm für diese bereits die passenden Antworten von Label-Vektoren mitgeben. Solch eine Menge von Wort- und zugehörenden Label-Vektoren werden auch *Trainingsdaten* genannt. Fragen wir es nun nach den Labels von einen dieser Wort-Vektoren, könnte es uns einfach den zugehörenden Label-Vektor orakeln.

Spannend ist die Frage, wie das Orakel Labels für Wort-Vektoren rät, von denen es die Lösung nicht kennt. Dafür leitet es zunächst aus den Trainingsdaten eine multivariate Wahrscheinlichkeitsverteilung $p(X, Y)$ ab, die angibt, wie wahrscheinlich zu einem Wort-Vektor X ein Label-Vektor Y gehört. Das Orakel geht dabei von dem *vereinfachten Modell* aus, dass jedes Label $y_i \in Y$ nur vom vorherigen Label $y_{i-1} \in Y$ und dem entsprechenden Wort $x_i \in X$ abhängt. Ist dies der Fall, kann $p(X, Y)$ durch eine Formel der Form von Formel (1) beschrieben werden.

$$\boldsymbol{p(X,Y) = \prod_{t=1}^{\#X} p(y_t|y_{t-1}) * p(y_t|x_t)} \quad (1)$$

Wahrscheinlichkeiten der Form von (1) können immer in die Form von Formel (2) überführt werden.

$$mit\ \Theta_{i,j} \in \mathbb{R}^+,\ L = alle\ moeglichen\ Labels,\ W = alle\ moeglichen\ Woerter,$$
$$X^* = alle\ moeglichen\ Woerter\text{-}Vektoren,\ Y^* = alle\ moeglichen\ Label\text{-}Vektoren,$$
$$und\ Z = \sum_{X \in X^*} \sum_{Y \in Y^*} \prod_{t=1}^{\#X} exp(\sum_{i,j \in L} \Theta_{i,j} * 1_{y_t=i} * 1_{y_{t-1}=j} + \sum_{i \in L} \sum_{o \in W} \mu_{i,o} * 1_{y_t=i} * 1_{x_t=o}),$$
$$\boldsymbol{p(X,Y) = \frac{1}{Z} \prod_{t=1}^{\#X} exp(\sum_{i,j \in L} \Theta_{i,j} * 1_{y_t=i} * 1_{y_{t-1}=j} + \sum_{i \in L} \sum_{o \in W} \mu_{i,o} * 1_{y_t=i} * 1_{x_t=o})} \quad (2)$$

$1_{Bedingung}$ ist dabei eine Funktion, die genau dann Eins ist, wenn die Bedingung erfüllt ist und ansonsten Null. Je öfter $y_t = i$ und $y_{t-1} = j$ gemeinsam in der Verteilung vorkommen, desto größer wird $\Theta_{i,j}$ sein. Kommen $y_t = i$ und $y_{t-1} = j$ gar nicht gemeinsam vor, gilt $\Theta_{i,j} * 1_{y_t=i} * 1_{y_{t-1}=j} = 0$. Gleiches gilt für $y_t = i$, $x_t = o$ und $\mu_{i,o}$. Die Thetas werden auch als *Gewichte* bezeichnet. Z bildet die Summe der Werte für alle möglichen Vorkommen von X und Y . Nach der Division durch Z ist somit sichergestellt, dass Formel (2) zwischen 0 und 1 liegt, sowie dass die Summe über alle möglichen Wahrscheinlichkeiten 1 ist, wie es für eine Wahrscheinlichkeitsverteilung der Fall sein muss.

Durch eine Abbildung der Indizies i , j und o auf k kann dies wiederum zu Formel (3) vereinfacht werden.

$$\boldsymbol{p(X,Y) = \frac{1}{Z} \prod_{t=1}^{\#X} exp(\sum_{k=1}^{K} \Theta_k * f_k(y_t, y_{t-1}, x_t))}, \quad (3)$$
$$mit\ Z = \sum_{X \in X^*} \sum_{Y \in Y^*} \prod_{t=1}^{\#X} exp(\sum_{k=1}^{K} \Theta_k * f_k(y_t, y_{t-1}, x_t))$$

Wie in Formel (4) zu sehen ist, kann jede multivariate Wahrscheinlichkeit in eine bedingte Wahrscheinlichkeit umgerechnet werden.

$$\boldsymbol{p(Y|X) = \frac{p(X,Y)}{\sum_{Y' \in Y^*} p(Y',X)}} \quad (4)$$

Eine nahe liegende Art zu Orakeln ist nun Formel (5). Nebenbei sei erwähnt, dass ein so antrainiertes Orkal als ein Hidden Markov Model (HMM) bezeichnet wird.

$$\boldsymbol{orakel(X) = argmax_Y(p(Y|X))} \quad (5)$$

Aufgrund des vereinfachten Modells, dass jedes Label y_i nur vom vorherigen Label y_{i-1} und dem entsprechenden Wort x_i abhängt, ist Formel (1) nur eine Annäherung an $p(X,Y)$ und somit sind auch alle weiteren Umformungen nur eine Annäherung. Ein CRF vereinfacht das Modell nun weiter, in dem es einige Θ_k und entsprechende f_k im Model weglässt. Durch das Weglassen von vielen kleinen Θ_k kann ein CRF die Berechnungszeit beschleunigen, wobei die berechnete Verteilung und somit das spätere Orakeln nur marginal verändert werden. Auch erlaubt ein CRF negative Werte für Θ_k . Der Intuition nach sagen diese aus, dass ein gemeinsames Vorkommen von X und Y sehr unwahrscheinlich ist und geben somit Strafpunkte für die $argmax_Y$ -Orakel-Funktion. Des Weiteren können in einem CRF sogenannte *custom feature functions* $f_{k'}(y_t, y_{t-1}, X)$ angegeben werden, die nicht nur vom entsprechenden Wort x_i sondern vom ganzen Wort-Vektor X abhängig sein können. Die $\Theta_{k'}$ zu diesen custom feature functions werden so berechnet, dass beim Orakeln der Trainings-Vektoren X die entsprechenden Label-

Vektoren Y möglichst genau geraten werden, unter der Annahme, dass das Ergebnis berechnet werden muss und nicht einfach nachgeschaut werden darf. Die Praxis zeigt, dass ein CRF mit guten custom feature functions besser orakelt als ein HMM. Typische custom feature functions wären z. B.:

- $f_{k'_1}(y_t, y_{t-1}, X) = 1_{x_t \; ist \; ein \; Nomen}$
- $f_{k'_2}(y_t, y_{t-1}, X) = 1_{x_t \; ist \; in \; einem \; Domaenen-spezifischen \; Woerterbuch}$
- $f_{k'_3}(y_t, y_{t-1}, X) = 1_{x_t \; ist \; in \; einem \; Domaenen-spezifischen \; Woerterbuch \; und \; x_{t-1} \; ist \; ein \; Nomen}$

Abschließend seien zu CRF noch eine überraschende und eine gute Bemerkung erwähnt. Die Überraschende ist folgende: Aufgrund der Vereinfachung des Models wie auch den custom feature functions ist es möglich, dass ein CRF ein Wort-Vektor, der in den Trainingsdaten enthalten war, falsch orakelt. Die gute Bemerkung ist, dass sich die erste Befürchtung eines Informatikers nicht bestätigt, dass die Funktion $orakel(X) = argmax_Y(p(Y|X)$ in Laufzeit von $O(\#L^{\#X})$ berechnet werden muss. Stattdessen kann sie durch dynamisches Programmieren in $O(\#L^2 * \#X)$ berechnet werden.

Implementierung der NYT

Die Rezepte der Kochseite der NYT haben alle eine Zutatenliste. Jede Zeile der Zutatenliste ist in einer Datenbank abgespeichert, aufgespalten nach der Zutat, einer Mengenangabe, der Mengeneinheit und Anderem. In dieser Datenbank sind mehr als 130.000 solcher Einträge, welche alle als Trainingsdaten für das CRF genutzt werden können. Die Zutat erhält dabei das Label *NAME*, die Mengenangabe *QTY* und die Einheit *UNIT*. Nach dem IOB2-Format (https://en.wikipedia.org/wiki/Inside_Outside_Beginning) erhält das erste Wort eines Labels den Prefix B-. Folgende Wörter erhalten den Prefix I-. Als CRF-Bibliothek verwendet die NYT CRF++ (https://taku910.github.io/crfpp/). Abb. 2 bis 4 zeigen alle von Greene in CRF++-Format angegebenen custom feature functions.

Abb. 2: Traininigsdaten mit Feature Labels

```
3/4         I1   L12   NoCAP   NoPAREN   B-QTY
pound       I2   L12   NoCAP   NoPAREN   OTHER
shiitake    I3   L12   NoCAP   NoPAREN   B-NAME
mushrooms   I4   L12   NoCAP   NoPAREN   I-NAME
,           I5   L12   NoCAP   NoPAREN   OTHER
stemmed     I6   L12   NoCAP   NoPAREN   B-COMMENT
and         I7   L12   NoCAP   NoPAREN   I-COMMENT
quartered   I8   L12   NoCAP   NoPAREN   I-COMMENT
```

Die Trainingsdaten in Abb. 2 enthalten neben dem Wort-Vektor (1. Spalte) und dem Label-Vektor (letzte Spalte) eine Reihe von Feature-Labels (2.-5. Spalte). Die 2. Spalte gibt zu jedem Wort den Index des Wortes im Satz an. Die 3. Spalte klassifiziert die Sätze nach der Anzahl ihrer Wörter in Vierer-Abständen. Sätze der Länge 0-3 werden als L0 klassifiziert, Sätze der Länge 4-7 als L4, usw. Die 3. Spalte gibt an, ob das jeweilige Wort mit einem großen Buchstaben anfängt und die 4. Spalte ob, das Wort im Satz innerhalb von Klammern steht.

Die vom CRF zu verwendenden Features werden durch das Template in Abb. 3 angegeben. Ein Template hat das Format U*ID*:%x[*Zeile*,*Spalte*]. Das B-Template entspricht der Wahrscheinlichkeit $p(y_t|y_{t-1})$, dass auf das Label y_{t-1} das Label y_t folgt. Abb. 4 zeigt die aus den Templates abgeleiteten Features, wenn dass aktuelle Wort in Abb. 2 „mushrooms“ ist.

Aus den Trainingsdaten mit den Feature-Labels und den Templates generiert CRF++ custom feature functions wie in Abb. 5 exemplarisch gezeigt ist.

Abb. 5: Beispiele für von CRF++ aus Abb. 2 und 3. extrahierte custom feature functions

U02:%[0,0]	$\rightarrow\ 1_{y_t=I-Name\ und\ x_t=mushrooms}$
U02:%[-2,4]	$\rightarrow\ 1_{y_t=I-Name\ und\ x_{t-2}\ 4.Spalte\ ist\ NoPAREN}$
U13:%x[0,0]/%x[0,2]	$\rightarrow\ 1_{y_t=I-Name\ and\ x_t=mushrooms\ und\ x_t\ 2.Spalte\ ist\ L12}$
B	$\rightarrow\ 1_{y_t=i\ und\ y_{t-1}=j}\ \forall\ i,j\ \in\ L$

Abb. 3:
Feature Templates

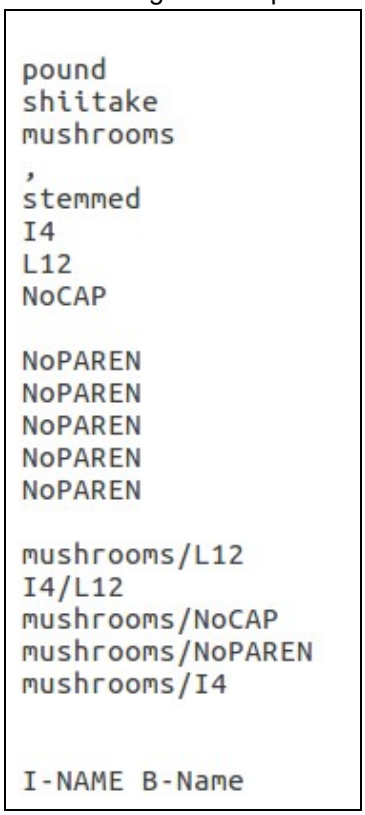
```
# Unigram
U00:%x[-2,0]
U01:%x[-1,0]
U02:%x[0,0]
U03:%x[1,0]
U04:%x[2,0]
U05:%x[0,1]
U06:%x[0,2]
U07:%x[0,3]

U08:%x[-2,4]
U09:%x[-1,4]
U10:%x[0,4]
U11:%x[1,4]
U12:%x[2,4]

U13:%x[0,0]/%x[0,2]
U14:%x[0,1]/%x[0,2]
U15:%x[0,0]/%x[0,3]
U16:%x[0,0]/%x[0,4]
U17:%x[0,0]/%x[0,1]

# Bigram
B
```

Abb. 4:
Anwendung der Templates



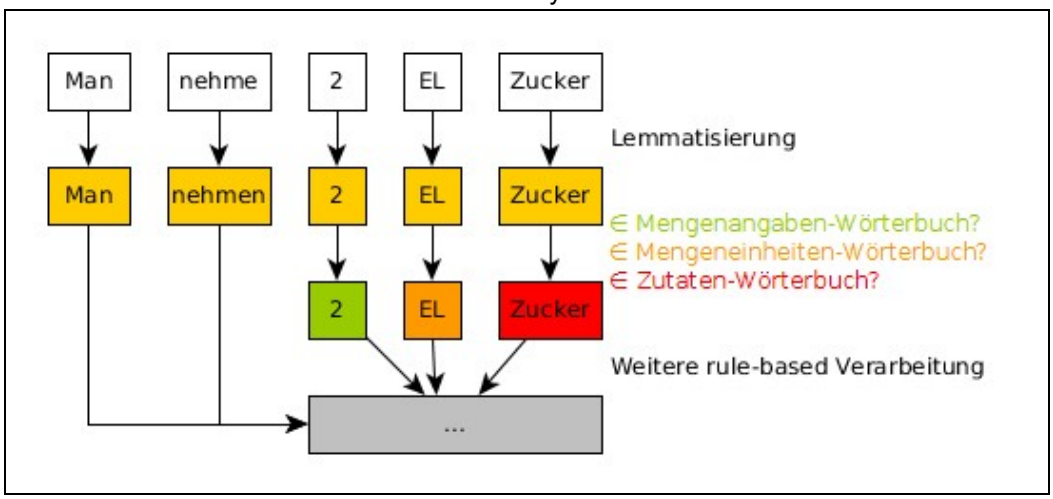
Diese CRF Implementierung berücksichtigt für ihre custom feature functions also nur die Position im Satz, die Länge des Satzes, ob ein Wort groß geschrieben wird und ob es eingeklammert ist. Bei einem Test mit 481 Rezepten orakelt dieses CRF 89% aller Zeilen einer Zutatenliste korrekt. (Greene, 2015) (Quellen.html#CRFNYT) testet nur 481 Rezepte, obwohl er über 130.000 zur Verfügung hat, da er keine Möglichkeit zur automatischen Auswertung gefunden hat. Wenn das CRF was anderes orakelt hat, als in der Datenbank stand, dann war manchmal das Orakelelte falsch, manchmal aber auch die manuell aufgebaute Datenbank und manchmal waren sogar beide falsch.

Dictionary- and rule-based

Die beiden unabhängigen Arbeiten von (Zeeshan, 2009) (Quellen.html#DictBased1) und (Wiegand et al., 2012) (Quellen.html#DictBased2) kategorisieren wir als *dictionary- and rule-based*. Die Grundidee ist in Abb. 6 zu sehen.

Sämtliche gesuchten *Entities* sind in einem Wörterbuch gespeichert. Da wir uns auf die Koch-Domäne beschränken, ist die Annahme der Existenz eines solchen Wörterbuches vertretbar. Für uns wäre dass z. B. ein Wörterbuch, welches alle Zutaten enthält, ein Weiteres für alle Mengenangaben und ein Drittes für alle Mengeneinheiten. Um diese Wörterbücher möglichst klein zu halten, werden nur die Lemmata der gesuchten Entities gespeichert. Um nun Informationen aus einem Satz zu extrahieren, wird der Satz zuerst lemmatisiert und die einzelnen Lemmata danach in den Wörterbüchern nachgeschaut. Ohne die Lemmatisierung könnte beispielsweise bei *„Man zerkocht die Äpfel“* Äpfel nicht extrahiert werden, obwohl *Apfel* im Wörterbuch steht. Die auf diese Weise extrahierten Entities können anschließend noch weiterverarbeitet werden, was wir als *rule-based* Weiterverarbeitung bezeichnen.

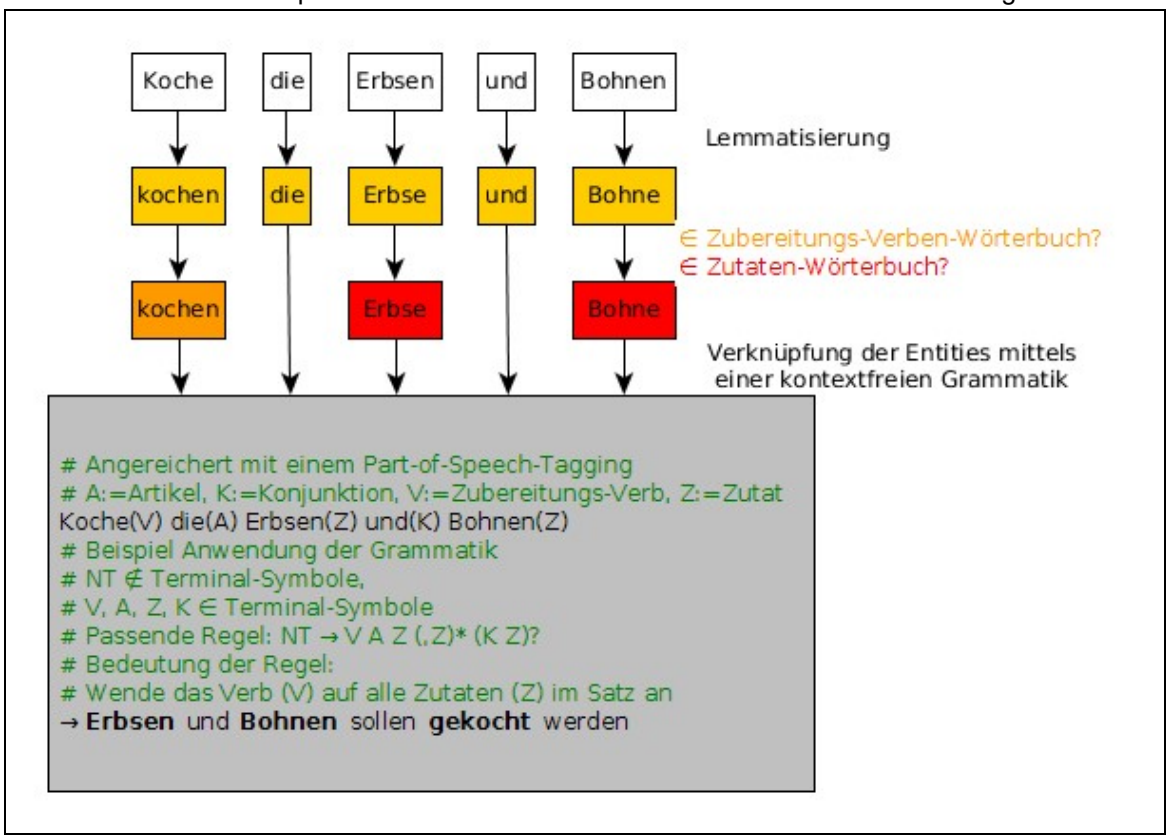
Abb. 6: Grundidee von dictionary- and rule-based extraction



(Zeeshan, 2009) (Quellen.html#DictBased1) möchte aus Rezepten die Zutaten extrahieren und herausfinden, wie diese verarbeitet werden. Zum Beispiel ob sie gekocht, oder gebacken werden. Nachdem er mit einem Wörterbuch von Zutaten und einem von Verarbeitungs-Verben die gesuchten Entities gefunden hat, muss er noch verbinden, welche Verarbeitungs-Verben zu welchen Zutaten gehören. Er hat dafür zwei unterschiedliche Ansätze:

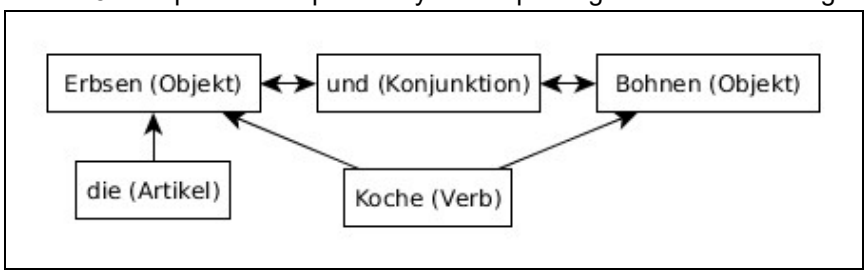
Der erste Ansatz verwendet eine **kontextfreie Grammatik**. Abb. 7 verdeutlicht das Prinzip beispielhaft. Die Terminal-Symbole der Grammatik sind Part-of-Speech-Tags sowie Tags für die bereits gefundenen Entities. Die Ableitungsregeln der Grammatik haben alle eine vordefinierte Bedeutung, wie z. B. *wende das Verarbeitungs-Verb auf alle Zutaten in diesen Satz an*. Ein Satz wird nun wie folgt bearbeitet: Zunächst wird er in Terminal-Symbole übersetzt. Anschließend wird geschaut, durch welche Regeln in der Grammatik diese Terminal-Symbole abgeleitet werden können. Wurde eine gültige Ableitung gefunden, wird abschließend die vordefinierte Bedeutung der Ableitung auf den Satz angewendet. Das Beispiel in Abb. 7 kommt so zu dem Ergebnis, dass bei dem Satz *Koche Erbsen und Bohnen* die *Erbsen* und die *Bohnen gekocht* werden sollen.

Abb. 7: Beispielhafte kontextfreie Grammatik-basierte Weiterverarbeitung



Der zweite Ansatz basiert auf **dependency-based parsing**. Abb. 8 zeigt ein exemplarisches Ergebnis eines dependency-based parsings des Satzes *Koche die Erbsen und Bohnen*. Aus diesem wird sofort ersichtlich, dass sich das Verb *kochen* auf die Objekte *Erbsen* und *Bohnen* bezieht. Das dependency-based parsing kann von Bibliotheken wie dem Stanford Parser (http://nlp.stanford.edu/software/lex-parser.shtml) erledigt werden.

Abb. 8: Beispielhafte dependency-based parsing Weiterverarbeitung



Beide Ansätze werden anhand von 43 Rezepten evaluiert. Tabelle 1 zeigt die Ergebnisse. Die Precision (Evaluierung.html#Metrics) zeigt, dass nahezu alle extrahierten Entities und ihre Beziehungen zueinander korrekt sind. Allerdings werden nur gut die Hälfte aller relevanten Informationen extrahiert, wie der Recall (Evaluierung.html#Metrics) zeigt. Die Schlussfolgerung von (Zeeshan, 2009) (Quellen.html#DictBased1) ist, dass das dictionary-based Extrahieren nahezu perfekt funktioniert. Wenn bei der rule-based Weiterverarbeitung eine Regel der kontextfreien Grammatik erfolgreich angewendet werden kann, sind die extrahierten Informationen ebenfalls korrekt. Allerdings sind viele Informationen nicht von einer Regel abgedeckt und werden somit nicht extrahiert. Gleiches gilt für das dependency-based parsing. Es werden zwar ein wenig mehr korrekte Informationen erfasst, dafür extrahiert dieser Ansatz auch einige zusätzliche Informationen, die falsch sind.

Tabelle 1

	Precision	**Recall**

Mit kontextfreier Grammatik	97,39%	51,54%
Mit dependency-based parsing	95,40%	64,12%

(Wiegand et al., 2012) (Quellen.html#DictBased2) sind an folgenden vier Fragen interessiert: *Welche Lebensmittel passen zu welchem Anlass?*, *Welche Lebensmittel werden oft kombiniert?*, *Welche Lebensmittel können untereinander ausgetauscht werden?* und *Welche Lebensmittel sind Teil eines Gerichts*. Um diese Fragen zu beantworten, extrahieren sie Informationen aus Wikipedia und dem Forum von Chefkoch.de. Sie arbeiten also mit rein textuellen Daten und können beispielsweise nicht auf der Struktur eines Rezeptes mit Zutatenliste aufbauen. Für uns relevant sind ihre Methoden *Surface Patterns* und *Statistical Co-occurrence*.

> **Surface Patterns** ist von der Idee her identisch mit der zuvor vorgestellten kontextfreien Grammatik von (Zeeshan, 2009) (Quellen.html#DictBased1) (Wiegand et al., 2012) (Quellen.html#DictBased2) stellen ebenfalls fest, dass die extrahierten Informationen meist korrekt sind, aber viele Informationen nicht gefunden werden. Im Gegensatz zu (Zeeshan, 2009) (Quellen.html#DictBased1) schreiben sie jedoch, dass sie mit einer dependency-based parsing Weiterverarbeitung nicht mehr korrekte Informationen extrahieren konnten. Sie führen das auf systematische Fehler bestehender Parser für die deutsche Sprache zurück. Diese sind mit Zeitungsartikel antrainiert, in welchen im Allgemeinen anders formuliert wird als in der Koch-Domäne.
>
> **Statistical Co-occurrence** sucht nach gemeinsamen Vorkommen von Entities. Wenn beispielsweise ein Lebensmittel (Popcorn) mit einem Anlass (Kinobesuch) im gleichen Text steht, wird extrahiert, dass das Lebensmittel zu diesem Anlass passt (Popcorn passt zum Kinobesuch).

Das Ergebnis der Extraktion ist je eine Liste von Lebensmittel für jede der vier Fragen. Da eine Liste von Ergebnissen nicht passend mit Percision und Recall ausgewertet werden kann und die Evaluierung keine weiteren Erkenntnisse für diese Arbeit bringt, sei für die Evaluierung auf die originale Ausarbeitung verwiesen.