6. Language Understanding (non-neural)

Ondřej Dušek & Vojtěch Hudeček & Jan Cuřín

http://ufal.cz/npfl123

24. 3. 2020
Natural Language Understanding

• **words → meaning**
  • whatever “meaning” is – can be different tasks
  • typically structured, explicit representation

• alternative names/close tasks:
  • *spoken language understanding*
  • *semantic decoding/parsing*

• integral part of dialogue systems, also explored elsewhere
  • stand-alone semantic parsers

• other applications:
  • human-robot interaction
  • question answering
  • machine translation (not so much nowadays)
NLU Challenges

• non-grammaticality

• disfluencies
  • hesitations – pauses, fillers, repetitions
  • fragments
  • self-repairs (~6%!)

• ASR errors

• synonymy

• out-of-domain utterances

find something cheap for kids should be allowed

uhm I want something in the west the west part of town
uhm find something uhm something cheap no I mean moderate
uhm I’m looking for a cheap

I’m looking for a for a chip Chinese rest or rant

Chinese city centre
uhm I’ve been wondering if you could find me
a restaurant that has Chinese food close to
the city centre please

oh yeah I’ve heard about that place my son was there last month
Semantic representations

- **syntax/semantic trees**
  - typical for standalone semantic parsing
  - different variations

- **frames**
  - technically also trees, but not directly connected to words
  - (mostly older) DSs, some standalone parsers

- **graphs (AMR)**
  - more of a toy task, but popular

- **dialogue acts** = intent + slots & values
  - flat – no hierarchy
  - most DSs nowadays

Examples:

- oui l’hôtel don’t le prix ne dépasse pas cent dix euros
- inform(date=Friday, stay=“2 nights”)
NLU basic approaches

For trees/frames/graphs:

- **grammar-based parsing**
  - handwritten/probabilistic grammars & chart parsing algorithms

- **statistical**
  - inducing structure using machine learning
  - grammar is implicit (training treebanks)

For DAs (shallow parsing):

- **classification**
- **sequence labelling**
Grammars vs. shallow parsing

Grammars are:

- more expressive
  - hierarchical structure better captures relations
- harder to maintain
  - sparser
  - harder to build rules by hand
  - statistical parsers need more data
  - training data is harder to get
- more hardware-hungry
  - chart parsing: $O(n^3)$, shallow: $O(n)$ for simplest approaches
- more brittle
  - shallow parsing is typically less sensitive to ASR errors, variation, etc.

Show me flights from Seattle to Boston

(Wang et al., 2005)
Grammars: CFG
(Context-free Grammar)

• Simple recursive grammar
  • **rules**: \( X \rightarrow A \ B \ C \)
    - splitting a phrase into adjacent parts
  • **terminals** = words
  • **non-terminals** = phrases (spanning multiple words)

• parsable using dynamic programming
  • (chart parsing)

• too simple for full natural language
  • but may be OK for a limited domain
  • especially with **probabilistic extensions**

\[
S \rightarrow NP \ VP | N \ VP | N \ V | NP \ V \\
VP \rightarrow V \ NP | V \ N | VP \ PP \\
NP \rightarrow D \ N | NP \ PP | N \ PP \\
PP \rightarrow P \ NP | P \ N
\]

\( N \rightarrow john, girl, car \) noun
\( V \rightarrow saw, walks \) verb
\( P \rightarrow in \) preposition
\( D \rightarrow the, a \) determiner

Alternative rules:
- ambiguous
CFG: Phoenix Parser
(ATIS, 90’s)

- CFG hierarchy based on **semantic frames**
  - Frames → slots / other frames
  - multiple CFGs, one per slot

- Robustness attempts
  - ignore stuff not belonging to any frame

- Chart parsing
  - left to right
  - maximize coverage
  - minimize # of different slots

\[\text{I would like to go to Boston tomorrow from San Francisco}\]
Grammars: CCG
(Combinatory Categorial Grammar)

• Grammar based on lambda calculus
  • syntax-bound semantics: lambda meaning in parallel to syntax phrases

• CCG lambda expressions:
  • logical constant: NYC, BOSTON…
  • variable: \( x, y, z \)…
  • literal: \( \text{city(AUSTIN), located_in(AUSTIN, TEXAS)} \)
  • lambda terms – binding variables: \( \lambda x. \text{city}(x) \sim \text{“}x \text{ is a city} \text{”} \)
  • quantifiers \( \exists \ \forall \), logical operators \( \Lambda \ \lor \ \neg \)

• CCG categories: syntax + lambda
  • simple: NOUN : \( \lambda x. \text{city}(x) \)
  • complex: \( S \backslash NP/NP : \lambda x. f(x) \) (“sentence missing an NP to the left and right”)

• Lexicon: word + syntax + lambda:
  • city \( \vdash \) NOUN: \( \lambda x. \text{city}(x) \), is \( \vdash \) S\backslash NP/NP : \( \lambda x. f(x) \)
Grammars: CCG

- parsing = combining categories (function application)
  - much fewer operations than CFG
    - $\Rightarrow, \langle \Rightarrow$ function application – $B : g + A \backslash B : f \Rightarrow A : f(g)$
    - $\Rightarrow B, \langle B$ function composition – $A/B : f + B/C : g \Rightarrow A/C : \lambda x.f(g(x))$
    - $\langle \Phi$ coordination (2 identical categories $\Rightarrow 1$)
  - $\neg$ category change
  - similar algorithms to CFG
  - statistical parsers available

I want to go from Boston to New York and then from New York to Chicago:

| CCG | NP
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$S/N$</td>
<td>$\lambda x. f(x)$</td>
</tr>
<tr>
<td>$\text{NP}$</td>
<td>$\text{BOS}$</td>
</tr>
<tr>
<td>$\text{I}$</td>
<td>$\text{NYC}$</td>
</tr>
</tbody>
</table>

$\Rightarrow_B$:

$\lambda f, \lambda x. f(x) \land \text{from}(x, \text{BOS}) \land \text{to}(x, \text{NYC})$

$\langle \Phi$:

$\lambda x, \lambda x[1]. f(x) \land \text{from}(x[1], \text{BOS}) \land \text{to}(x[1], \text{NYC}) \land \text{before}(x[1], x[2]) \land \text{to}(x[2], \text{CHI})$


https://yoavartzi.com/tutorial/
NLU as classification

• using DAs – treating them as a set of semantic concepts
  • concepts:
    • intent
    • slot-value pair
  • binary classification: is concept Y contained in utterance X?
  • independent for each concept

• consistency problems
  • no conflicting intents (e.g. affirm + negate)
  • no conflicting values (e.g. kids-allowed=yes + kids-allowed=no)
  • need to be solved externally, e.g. based on classifier confidence
NLU as classification

- **classification:** features $\rightarrow$ labels (classes)
  - here: classes are **binary** (-1/1 or 0/1)
  - **one classifier per concept**
- **features**
  - **binary** – is X present? or **count** – how many X’s are present?
  - words
  - n-grams
  - word pairs/triples (position-independent)
  - regex
  - presence of named entities

*I’m looking for something cheap in the city centre.*

*Dialogue act types*:
- negate
- deny
- inform
- select

*Slot value pairs*:
- food=Italian
- food=Chinese
- area=centre
- area=north
- price=cheap

(from Milica Gašić’s slides)
NER + delexicalization

Approach:

1) **identify** slot values/named entities

2) **delexicalize** = replace them with placeholders (indicating entity type)
   - or add the NE tags as more features for classification

- generally needed for NLU as classification
  - otherwise in-domain data is too sparse
  - this can vastly reduce the number of concepts to classify & classifiers

- NER is a problem on its own
  - but general-domain NER tools may need to be adapted
    - added gazetteers with in-domain names
  - in-domain gazetteers alone may be enough
  - NE supplemented by NE linking/disambiguation (usually not needed in DS)

What is the phone number for Golden Dragon?
What is the phone number for <restaurant-name>?

I’m looking for a Japanese restaurant in Notting Hill.
I’m looking for a <food> restaurant in <area>.
NLU Classifiers

• note that data is usually scarce!

• **handcrafted / rules**
  • simple mapping: word/n-gram/regex match → concept
  • can work really well for a limited domain
  • no training data, no retraining needed (tweaking on the go)

• **logistic regression**

• **SVM** (support vector machine)

• **neural nets**
  • different, “automatic” features (embeddings, see later)
  • only applicable if a lot of data is available
Logistic Regression
(Maximum Entropy Classifier)

- despite the name, it’s a classifier
- very basic, but powerful with the right features
- trained by gradient descent (logistic/cross entropy loss)
- maximum entropy estimate (“most uniform model given data”)

\[ p(y|x) = \frac{1}{1 + \exp(-y(\theta \cdot x))} \]

binary, for \( y \in \{-1, +1\} \)

\[ p(y|x) = \frac{1}{Z(x)} \exp(\theta \cdot f(x, y)) \]

normalization

generalization: feature functions vector
(some fire for each value of \( y \))

equivalent form
- maximum entropy style
(works for \textbf{multiclass}, too!)
Support-Vector Machines (SVMs)

• separate classes with **maximum margin** (=best generalization)
• decision boundary defined by **support vectors** (closest instances)

There are many possible separation boundaries between classes in feature space.

- boundary farthest away from both classes = maximum margin
- instances closest to the boundary = support vectors
- removing a support vector changes the boundary

(from Aikaterini Tzompanaki's slides)
SVMs

- Decision boundary: $\mathbf{\theta} \cdot \mathbf{x}^{\text{bound}} = 0$
- Support vectors: $\mathbf{\theta} \cdot \mathbf{x}^{sv} = y^{sv}$ ($y^{sv} \in \{-1, +1\}$)
- Maximum margin: $\max \frac{2}{||\mathbf{\theta}||} \sim \min \frac{1}{2} ||\mathbf{\theta}||^2$ with correct classification
  
  - constrained optimization – quadratic programming (Lagrange multipliers)

- SVM Score: $g(\mathbf{x}) = \mathbf{\theta} \cdot \mathbf{x} = \sum_{i=1}^{S} y_i \alpha_i \mathbf{x}_i \cdot \mathbf{x}$
- classification:
  - $y = \text{sign}(g(\mathbf{x}))$
- probability: Platt scaling
  - logistic regression with $g(\mathbf{x})$ as feature

why margin is $\frac{2}{||\mathbf{\theta}||}$: https://math.stackexchange.com/questions/1305925/
SVM vs. Logistic Regression

• **soft-margin SVM** – for non-separable cases
  • non-separable = no perfect decision boundary
  • “soft” = weighing correct classification *(hinge loss)* & margin size
  • model: \( \min_{\theta} \lambda ||\theta||^2 + \sum_i \max\{0, 1 - y_i \theta \cdot x_i\} \)

• **regularized logistic regression** – for better generalization
  • preventing overfitting to training data – trying to keep parameter values low
  • logistic loss
  • model: \( \min_{\theta} \lambda ||\theta||^2 + \sum_i \log(1 + \exp(1 - y_i \theta \cdot x_i)) \)

• the main difference is the loss
  • hinge loss should be marginally better for classification, but it depends
Classification example

**features** $(x)$

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>1</td>
</tr>
<tr>
<td>want</td>
<td>1</td>
</tr>
<tr>
<td>to</td>
<td>3</td>
</tr>
<tr>
<td>go</td>
<td>1</td>
</tr>
<tr>
<td>from</td>
<td>2</td>
</tr>
<tr>
<td>&lt;airport-1&gt;</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>him</td>
<td>0</td>
</tr>
<tr>
<td>price</td>
<td>0</td>
</tr>
<tr>
<td>tell</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>I want</td>
<td>1</td>
</tr>
<tr>
<td>want to</td>
<td>1</td>
</tr>
<tr>
<td>to go</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>from &lt;airport-1&gt;</td>
<td>1</td>
</tr>
</tbody>
</table>

**weights:**

- intent=search_flights
- intent=request_price
- from_airport=<airport-1>

weights define different classifiers

**ASR:** *I want to go from Newark to London City next Friday*

**Delex:** *I want to go from <airport-1> to <airport-2> next <day-1>*

**SVM:** $\theta_{FA1} \cdot x = +3.4347 \rightarrow$ found from_airport=Newark

**LR:** $\text{sigmoid}(\theta_{FA1} \cdot x) = 0.883 \rightarrow$ found from_airport=Newark (conf. = 0.883)
Slot filling as sequence tagging

- get slot values directly – “automatic” delexicalization
  - each word classified
  - classes = slots & IOB format (inside-outside-beginning)
  - slot values taken from the text (where a slot is tagged)
  - NER-like approach

- rules + classifiers kinda still work
  a) keywords/regexes found at specific position
  b) apply classifier to each word in the sentence left-to-right
  - problem: overall consistency
    - slots found elsewhere in the sentence might influence what’s classified now

- solution: structured/sequence prediction
Maximum Entropy Markov Model (MEMM)

- Looking at past classifications when making next ones
  - LR + a simple addition to the feature set
- Whole history would be too sparse/complex
  \[ \rightarrow \textbf{Markov assumption}: \] only the most recent matters
  - 1\textsuperscript{st} order MM: just the last one (\[\leftarrow\text{this is what we show here}\])
  - \(n\)\textsuperscript{th} order MM: \(n\) most recent ones
- still not modelling the sequence globally

\[
p(y|x) = \prod_{t=1}^{T} \frac{1}{Z(y_{t-1}, x)} \exp(\theta \cdot f(y_t, y_{t-1}, x))
\]

- \(p(y|x)\) for the whole sequence
- time steps – independent except for \(y_{t-1}\)
- \(y_{t-1}\) is the main addition compared to LR
- looking at the whole input
Hidden Markov Model (HMM)

- Modelling the **sequence as a whole**
- Very basic model:
  - “tag depends on word + previous tag”
- Markov assumption, again
- “Hidden” – reverse viewpoint:
  - “tags are hidden, but they influence the words on the surface”
- Inference – Viterbi algorithm
  - we can get the **globally best tagging**

HMM is a **generative model** – models **joint distribution** $p(y, x)$, not just conditional $p(y|x)$

$$p(y, x) = \prod_{t=1}^{T} p(y_t|y_{t-1})p(x_t|y_t)$$

for the whole sequence

transition probability prev. tag $\rightarrow$ tag

observation probability tag $\rightarrow$ word
Hidden Markov Model

• Rewrite so it looks more like MEMM + get conditional probability

Just indicators (binary features)

$$p(y, x) = \prod_{t=1}^{T} \exp\left( \sum_{i,j \in S} \theta_{ij} 1_{y_t=i} 1_{y_{t-1}=j} + \sum_{i \in S} \sum_{o \in O} \mu_{oi} 1_{y_t=i} 1_{x_t=o} \right)$$

Transition

Observation

Hide the actual probabilities as weights (in logarithm)

Subsume transition & observation under feature functions, $\theta_k$ is $\theta_{ij}$ & $\mu_{oi}$

Just the current word

Conditional probability:

$$p(y|x) = \frac{p(y, x)}{\sum_{y'} p(y', x)} = \frac{1}{Z(x)} \prod_{t=1}^{T} \exp\left( \sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, x_t) \right) = \frac{1}{Z(x)} \prod_{t=1}^{T} \exp(\theta \cdot f(y_t, y_{t-1}, x_t))$$

Normalization is global

Vector notation
HMM vs. MEMM

• MEMM:
  • any feature functions, as in LR
  • local normalization – does not model whole sequences, just locally
  • label bias problem
    • training: you know the correct labels
    • inference: one error can lead to a series of errors

• HMM:
  • global normalization for $p(y|x)$ over all $y$’s
    • modelling sequences as a whole
  • very boring & limited feature functions

• how about best of both?
Linear-Chain
Conditional Random Field (CRF)

- HMM + more complex feature functions
- MEMM + global sequence modelling

\[ p(y|x) = \frac{1}{Z(x)} \prod_{t=1}^{T} \exp(\theta \cdot f(y_t, y_{t-1}, x)) \]

- state-of-the-art for many sequence tagging tasks (incl. NLU)
  - until NNs took over
  - used also in conjunction with NNs
- global normalization makes it slow to train
Sequence tagging example

ASR:  
*I want to go from from Newark*  
to London City next Friday

Previous tags:  
0 0 0 0 0 B-from_airport 0

current position:  
what’s the class for London?

features (x):

<table>
<thead>
<tr>
<th>in_sent=</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newarke</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>want</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>to</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>go</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>him</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>price</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>I want</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>want to</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>to go</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

features (x):

| cur=London | 1 | prev_tag=O | 1 |
| cur=him    | 0 | prev_tag=B-price | 0 |
| cur=to     | 1 |   |   |
| prev=to    | 1 |   |   |
| prev=want  | 0 |   |   |
| prev=price | 0 |   |   |
| cur=to London | 1 |   |   |
| prev=Newark to | 1 |   |   |

HMM considers only these

MEMM: looks at London, ignores that it also needs to tag City later  
→ likely to tag as B-to_city

CRF: also considers future tags, more likely to tag London City  
as B-to_airport I-to_airport
Handling ASR noise

• ASR produces multiple hypotheses
• Combine & get resulting NLU hypotheses
  • NLU: $p(DA|text)$
  • ASR: $p(text|audio)$
  • we want $p(DA|audio)$
• Easiest: **sum it up**

$$p(DA|audio) = \sum_{texts} P(DA|text)P(text|audio)$$

0.33 – I am looking for a bar
0.26 – I am looking for the bar
0.11 – I am looking for a car
0.09 – I am looking for the car
0.59 – inform(task=find, venue=bar)
0.20 – null()

(from Filip Jurčiček’s slides)
Handling ASR noise

• Alternative: **use confusion networks**
  • per-word ASR confidence
  • Word features weighed by word confidence

---

0.33 — I am looking for a bar  
0.26 — I am looking for the bar  
0.11 — I am looking for a car  
0.09 — I am looking for the car

~equivalent confusion network

**features:**

- I                0.9
- hi               0.02
- am               0.9
- looking          1
- for              1
- ...             
- I am             0.81
- my am            0.063
- am looking       0.9
- a bar            0.3
- a car            0.24
- ...             

should be normalized by n-gram length

(from Filip Jurčiček’s slides)
Context

• user response can depend on last system action
  • fragments/short replies are ambiguous without context
• add last system DA/text into input features
  • helps disambiguate
• careful – user may not play nice!
  • system needs to be trained with both alternatives in mind

U: I’m looking for flights from JFK.
S: Where would you like to go?
U: Atlanta.

inform(??=Atlanta)
inform(from=Atlanta)

x U: Actually I’d rather fly from Newark.
Summary

• NLU can be tricky
  • bad grammar, fragments, synonymy, ASR errors …

• Grammars, frames, graph representation
  • rule-based or statistical structure induction
  • more expressive, but harder – not so much in limited-domain systems

• Shallow parsing
  • dialogue acts: intent + slots & labels
  • rules – keyword spotting, regex
  • classification (LR, SVM)
  • sequence tagging (MEMM, HMM, CRF)

• Next time: neural NLU & dialogue state tracking

(Sutton & McCallum, 2010)
https://arxiv.org/abs/1011.4088
Thanks

Contact us:
odusek@ufal.mff.cuni.cz
hudecek@ufal.mff.cuni.cz
Slack

Get these slides here:
http://ufal.cz/npfl123

References/Inspiration/Further:

• Milica Gašić’s slides (Cambridge University): http://mi.eng.cam.ac.uk/~mg436/teaching.html
• Raymond Mooney’s slides (University of Texas Austin): https://www.cs.utexas.edu/~mooney/ir-course/
• Hao Fang’s slides (University of Washington): https://hao-fang.github.io/ee596_spr2018/syllabus.html
• Aikaterini Tzompanaki’s slides (University of Cergy-Pontoise): https://perso-etis.ensea.fr/tzompanaki/teaching.html
• Pierre Lison’s slides (University of Oslo): https://www.uio.no/studier/emner/matnat/ifi/INF5820/h14/
• Andrew McCallum’s slides (U. of Massatchusets Amherst): https://people.cs.umass.edu/~mccallum/courses/inlp2007/