Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorial Grammars

Luke S. Zettlemoyer and Michael Collins

MIT CSAIL

UAI 2005

semantic parsing

- **input**: sentence
- **output**: logical form
- dataset
  - Geo880, queries to a database of United States geography
  - Jobs640, queries to a database of job listings

Example: What states border Texas?

$$\lambda x.\mathrm{state}(x) \wedge \mathrm{borders}(x,\mathrm{texas})$$

lambda calculus

- constants: entities, numbers, functions
- logical connectives: conjunction (∧), disjunction (∨), negation(¬), implication (→)
- quantifiers: universal (∀) and existential (∃)
- lambda expressions: anonymous functions (λx.f (x))
- other quantifiers/functions: arg max, definite descriptions

Example: What states border Texas?

$\lambda x.state(x) \wedge borders(x,texas)$

lambda calculus

- constants: entities, numbers, functions
- logical connectives: conjunction ($\wedge$), disjunction ($\vee$), negation($\neg$), implication ($\rightarrow$)
- quantifiers: universal ($\forall$) and existential ($\exists$)
- lambda expressions: anonymous functions ($\lambda x.f(x)$)
- other quantifiers/functions: arg max, definite descriptions

Example: What is the largest state?

$$\arg\max(\lambda x.state(x), \lambda x.size(x))$$

CCG

- **application rules**

$$A/B : f \qquad B : g \Rightarrow A : f(g)$$

- **composition rules**

$$A/B : f \qquad B/C : g \Rightarrow A/C : \lambda x.f(g(x))$$

- **type-raising rules**

$$NP : f \Rightarrow NP/(S\backslash NP) : \lambda g.g(f)$$

$$\cfrac{\cfrac{\text{Utah}}{NP} \qquad \cfrac{\cfrac{\text{borders}}{(S\backslash NP)/NP} \qquad \cfrac{\text{Idaho}}{NP}}{S\backslash NP}\,>}{S}\,<$$

Utah := NP : utah

Idaho := NP : idaho

borders := $(S\backslash NP)/NP : \lambda x.\lambda y.\text{borders}(y, x)$

CCG

Utah	borders	Idaho
NP	(S\NP)/NP	NP
utah	*λx.λy.borders(y, x)*	*Idaho*

$>$

S\NP

λy.borders(y, idaho)

$<$

S

borders(utah, idaho)

main idea

- **Probabilistic Model:**
  - Deciding between different analyses, handling spurious ambiguity.
- **Lexical Extraction:**
  - Based on template

log-linear model

- **Log-Linear Model**
  - A set X of inputs (e.g. sentences)
  - A set Y of labels/structures.
  - A feature function $f : X \times Y \rightarrow R^d$ for any pair (x,y)
- **Conditional Model**

$$p(y \mid x; \theta) = \frac{e^{f(x,y) \cdot \theta}}{Z(x, \theta)}$$

  - inner product

$$f(x, y) \cdot \theta = \sum_{k=1}^{d} \theta_k f_k(x, y)$$

  - normalization term

$$Z(x, \theta) = \sum_{y' \in \mathcal{Y}} e^{f(x,y') \cdot \theta}$$

log-linear model

- **Maximum Likelihood:** find a model θ* that maximizes LCL

$$\theta^* = \max_\theta \prod_{i=1}^{n} p(y_i \mid x_i; \theta)$$

$$= \max_\theta \sum_{i=1}^{n} log\ p(y_i \mid x_i; \theta)$$

- **logarithm of conditional likelihood (LCL)**

$$\mathcal{O}(\theta) = \sum_{i=1}^{n} log\ p(y_i \mid x_i; \theta)$$

- **Computing Gradient**

$$\frac{\partial}{\partial \theta_j} log\ p(y \mid x; \theta) = \sum_{i=1}^{n} f_j(x_i, y_n) - \sum_{i=1}^{n} \sum_{y' \in \mathcal{Y}} p(y \mid x_i; \theta) f_j(x_i, y)$$

log-linear model

- **Maximum Likelihood:** find a model θ* that maximizes LCL

$$\theta^* = \max_\theta \prod_{i=1}^{n} p(y_i \mid x_i; \theta)$$

$$= \max_\theta \sum_{i=1}^{n} log\ p(y_i \mid x_i; \theta)$$

- **logarithm of conditional likelihood (LCL)**

$$\mathcal{O}(\theta) = \sum_{i=1}^{n} log\ p(y_i \mid x_i; \theta)$$

- **Computing Gradient**

$$\frac{\partial}{\partial \theta_j} log\ p(y \mid x; \theta) = \boxed{\sum_{i=1}^{n} f_j(x_i, y_n)} - \boxed{\sum_{i=1}^{n} \sum_{y' \in \mathcal{Y}} p(y \mid x_i; \theta) f_j(x_i, y)}$$

Empirical counts

Expected counts

Probabilistic CCGs

- deciding between different analyses, handling spurious ambiguity
  - lexical items have more than one entry
  - logical form L may be derived by multiple derivations T
- Use a log-linear formulation of CCG (Clark and Curran (2003))

$$p(L, T \mid S; \theta) = \frac{e^{f(L,T,S)\cdot\theta}}{\sum_{(L,T)} e^{f(L,T,S)\cdot\theta}}$$

- Parsing Problem

$$\underset{L}{\operatorname{argmax}} P(L \mid S; \theta) = \underset{L}{\operatorname{argmax}} \sum_{T} p(L, T \mid S; \theta)$$

  - note: T might be very large, use dynamic programming.

Probabilistic CCGs

- **Local features**
  - Limit features to lexical rules in derivations
- Why?
  - for convenience of dynamic programming

$$
\dfrac{\dfrac{\text{Oklahoma}}{\texttt{NP : oklahoma'}} \quad \dfrac{\dfrac{\text{borders}}{\texttt{(S\textbackslash NP)/NP : } \lambda y, \lambda x.borders(x,y)} \quad \dfrac{\text{Texas}}{\texttt{NP : texas'}}}{\texttt{S\textbackslash NP : } \lambda x.borders(x, texas')}(>)}{\texttt{S : borders(oklahoma',texas')}}(<)
$$

$$
f_{id(\texttt{NP : texas'})}(x, y) = count(\texttt{NP : texas'}) = 1
$$

Probabilistic CCGs

- **LCL objective**

$$\mathcal{O}(\theta) = \sum_{i=1}^{n} log\ p(L_i \mid x_i; \theta) = \sum_{i=1}^{n} log\Big(\sum_{T} p(L_i, T \mid x_i; \theta)\Big)$$

- **Computing Gradient**

$$\frac{\partial}{\partial \theta_j}\mathcal{O}(\theta) = \sum_{i=1}^{n}\sum_{T} f_j(L_i, T, x_i) p(T \mid L_i, x_i; \theta) - \sum_{i=1}^{n}\sum_{L,T} f_j(L, T, x_i) p(L, T \mid x_i; \theta)$$

  - note: This involves find the probability of all trees/derivations and their features given an input
  - Dynamic programming: Use variant of inside-outside algorithm

GenLex: Lexical extraction

- **GenLex:** Take a sentence and logical form and generates lexical items.

$$\textsc{GenLex}(\mathrm{S},\mathrm{L}) = \{x := y \mid x \in W(S), y \in C(L)\}$$

  - W(S): set of substrings in input S
  - C(L): CCG rule templates or triggers

GenLex: Lexical extraction

Rules		Categories produced from logical form $\arg\max(\lambda x.state(x) \wedge borders(x, texas), \lambda x.size(x))$
Input Trigger	Output Category	
constant c	$NP : c$	$NP : texas$
arity one predicate p_1	$N : \lambda x.p_1(x)$	$N : \lambda x.state(x)$
arity one predicate p_1	$S\backslash NP : \lambda x.p_1(x)$	$S\backslash NP : \lambda x.state(x)$
arity two predicate p_2	$(S\backslash NP)/NP : \lambda x.\lambda y.p_2(y, x)$	$(S\backslash NP)/NP : \lambda x.\lambda y.borders(y, x)$
arity two predicate p_2	$(S\backslash NP)/NP : \lambda x.\lambda y.p_2(x, y)$	$(S\backslash NP)/NP : \lambda x.\lambda y.borders(x, y)$
arity one predicate p_1	$N/N : \lambda g.\lambda x.p_1(x) \wedge g(x)$	$N/N : \lambda g.\lambda x.state(x) \wedge g(x)$
literal with arity two predicate p_2 and constant second argument c	$N/N : \lambda g.\lambda x.p_2(x, c) \wedge g(x)$	$N/N : \lambda g.\lambda x.borders(x, texas) \wedge g(x)$
arity two predicate p_2	$(N\backslash N)/NP : \lambda x.\lambda g.\lambda y.p_2(x, y) \wedge g(x)$	$(N\backslash N)/NP : \lambda g.\lambda x.\lambda y.borders(x, y) \wedge g(x)$
an $\arg\max$ / $\min$ with second argument arity one function f	$NP/N : \lambda g.\arg\max / \min(g, \lambda x.f(x))$	$NP/N : \lambda g.\arg\max(g, \lambda x.size(x))$
an arity one numeric-ranged function f	$S/NP : \lambda x.f(x)$	$S/NP : \lambda x.size(x)$

GenLex: Lexical extraction

Example: Oklahoma borders Texas.
borders(oklahoma,texas)

- W (Oklahoma borders Texas) = {”Oklahoma”, ”Texas”,”Oklahoma borders”, ...}
- C(borders(oklahoma, texas)) = {borders(...) $\rightarrow$ (S\NP)/NP : $\lambda y, \lambda x.$borders(x, y); texas $\rightarrow$ NP : texas, ...}

Learning Algorithm

- **learning**
  - lexical generation
  - log-linear parameter estimation
- **big idea**
  - Learn compact lexicons via greedy iterative method that works with high probability rules/derivations

Learning Algorithm

- an initial lexicon, Λ_0
  - includes lexical items that are derived directly from the database in the domain
  - such as { Utah := NP : utah, Idaho := NP : idaho, Nevada := NP : nevada, . . .}
- possible lexical items set

$$\Lambda^* = \Lambda_0 \cup \bigcup_{i=1}^{n} \mathrm{GENLEX}(S_i, L_i)$$

  - parameter initialization: 0.1 for all lexical items in Λ_0, and 0.01 for all other lexical items

Learning Algorithm

Algorithm:

- For $t = 1 \dots T$

Step 1: (Lexical generation)

- For $i = 1 \dots n$:
  - Set $\lambda = \Lambda_0 \cup \text{GENLEX}(S_i, L_i)$.
  - Calculate $\pi = \boxed{\text{PARSE}(S_i, L_i, \lambda, \bar{\theta}^{t-1})}$.
  - Define λ_i to be the set of lexical entries in π.
- Set $\Lambda_t = \Lambda_0 \cup \bigcup_{i=1}^{n} \lambda_i$

Step 2: (Parameter Estimation)

- Set $\bar{\theta}^t = \boxed{\text{ESTIMATE}(\Lambda_t, E, \bar{\theta}^{t-1})}$

Output: Lexicon Λ_T together with parameters $\bar{\theta}^T$.

Returns the highest probability parse for S_i with logical form L_i

Returns parameter values θ that are the output of SGD

- **Step 1**: Search for small set of lexical entries to parse data, then parse and find most probable rules.
- **Step 2**: Re-estimate log-linear model based on these compact lexical entries.

Results

	Geo880		Jobs640	
	P	R	P	R
Our Method	96.25	79.29	97.36	79.29
COCKTAIL	89.92	79.40	93.25	79.84

- Geo880(600/280), Jobs640(500/140)
- Creates compact lexicons
- Easily inject linguistic knowledge
- Weakly supervised learning

Inducing Probabilistic CCG Grammars from Logical Form with Higher-Order Unification

Tom Kwiatkowski*, Luke Zettlemoyer, Sharon Goldwater and Mark Steedman

University of Edinburgh and University of Washington

EMNLP 2010

Unification-based GENLEX

- Automatically learns the templates
  - Can be applied to any language and many different approaches for semantic modeling
- Two step process
  - Initialize lexicon with labeled logical forms
  - “Reverse” parsing operations to split lexical entries

Unification-based GENLEX

- Initialize lexicon with labeled logical forms

For every labeled training example:

I want a flight to Boston

$\lambda x.flight(x) \wedge to(x, BOS)$

Initialize the lexicon with:

I want a flight to Boston $\vdash$ S : $\lambda x.flight(x) \wedge to(x, BOS)$

Splitting lexical entries

$$\text{I want a flight to Boston} \vdash S : \lambda x.flight(x) \land to(x, BOS)$$

$$\text{I want a flight} \vdash S/(S|NP) : \lambda f.\lambda x.flight(x) \land f(x)$$

$$\text{to Boston} \vdash S|NP : \lambda x.to(x, BOS)$$

Many possible phrase pairs × Many possible category pairs

$$\mathrm{S}_L(w_{0:n} \vdash A) = \{(w_{0:i} \vdash B, w_{i+1:n} \vdash C) \mid 0 \leq i < n \land (B, C) \in \mathrm{S}_C(A)\}$$

Splitting lexical entries

- Split logical form h to f and g s.t.

$$f(g) = h \text{ or } \lambda x.f(g(x)) = h$$

- Infer syntax from logical form type
  - C(T) takes an input type T and returns the syntactic category of T

$$C(T) = \begin{cases} NP & \text{if } T = e \\ S & \text{if } T = t \\ C(T_2)|C(T_1) & \text{when } T = \langle T_1, T_2 \rangle \end{cases}$$

eg. C(<e, t>) = S|NP   C(<e, <e, t>>) = S|NP|NP

T(tex) = e   T(λx.state(x)) = <e, t>

Splitting lexical entries

$$S : \lambda x.flight(x) \wedge to(x, BOS)$$

$$\lambda f.\lambda x.flight(x) \wedge f(x)$$
$$\lambda x.to(x, BOS)$$

$$\lambda y.\lambda x.flight(x) \wedge f(x, y)$$
$$BOS$$
$$\ldots$$

Splitting lexical entries

$$S : \lambda x.flight(x) \wedge to(x, BOS) \Rightarrow$$

$$\begin{aligned}
\boxed{S/(S|NP)} &: \lambda f.\lambda x.flight(x) \wedge f(x) \\
\boxed{S|NP} &: \lambda x.to(x, BOS) \\
\\
\boxed{S/NP} &: \lambda y.\lambda x.flight(x) \wedge f(x, y) \\
\boxed{NP} &: BOS \\
&\ldots
\end{aligned}$$

Splitting lexical entries

- Split logical form h to f and g

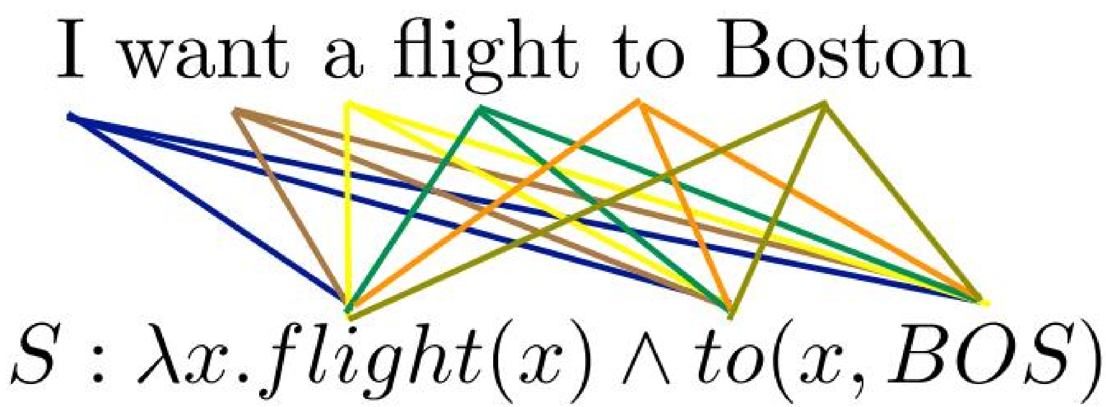
$$\mathrm{FA}(X{:}h) = \{(X/Y{:}f, Y{:}g) \mid h{=}f(g) \wedge Y{=}C(T(g))\}$$

$$\begin{aligned}\mathrm{FC}(X/Y{:}h) = \{(&X/W{:}f, W/Y{:}g) \mid \\ &h{=}\lambda x.f(g(x)) \wedge W{=}C(T(g(x)))\}\end{aligned}$$

$$\mathrm{S}_C(A) = \{\mathrm{FA}(A) \cup \mathrm{BA}(A) \cup \mathrm{FC}(A) \cup \mathrm{BC}(A)\}$$

Parameter Initialization

- Compute co-occurrence (IBM Model 1) between words and logical constants



- Initial score for new lexical entries: average over pairwise weights

Learning Algorithm

I want a flight to Boston

$\lambda x.flight(x) \wedge to(x, BOS)$

1. Find highest scoring correct parse
2. Find splits that most increases score
3. Return new lexical entries

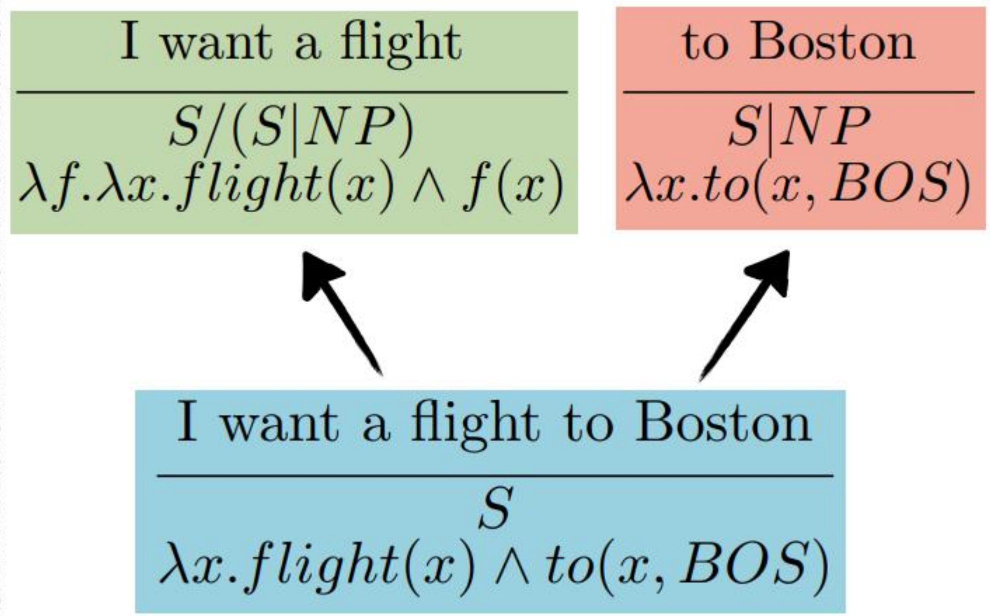
$$\frac{\text{I want a flight to Boston}}{\begin{array}{c} S \\ \lambda x.flight(x) \wedge to(x, BOS) \end{array}}$$

Learning Algorithm

I want a flight to Boston

$\lambda x.flight(x) \land to(x, BOS)$

1. Find highest scoring correct parse
2. Find splits that most increases score
3. Return new lexical entries



Learning Algorithm

I want a flight to Boston

$\lambda x.flight(x) \wedge to(x, BOS)$

1. Find highest scoring correct parse
2. Find splits that most increases score
3. Return new lexical entries

$$\frac{\text{I want a flight}}{\begin{array}{c} S/(S|NP) \\ \lambda f.\lambda x.flight(x) \wedge f(x) \end{array}} \qquad \frac{\text{to Boston}}{\begin{array}{c} S|NP \\ \lambda x.to(x, BOS) \end{array}}$$

$$\frac{\text{I want a flight to Boston}}{\begin{array}{c} S \\ \lambda x.flight(x) \wedge to(x, BOS) \end{array}}$$

Learning Algorithm

I want a flight to Boston

$\lambda x.flight(x) \wedge to(x, BOS)$

1. Find highest scoring correct parse
2. Find splits that most increases score
3. Return new lexical entries

Iteration 2

$$
\dfrac{\dfrac{\text{I want a flight}}{\begin{array}{c} S/(S|NP) \\ \lambda f.\lambda x.flight(x) \wedge f(x) \end{array}} \quad \dfrac{\text{to Boston}}{\begin{array}{c} S|NP \\ \lambda x.to(x, BOS) \end{array}}}{\begin{array}{c} S \\ \lambda x.flight(x) \wedge to(x, BOS) \end{array}} >
$$

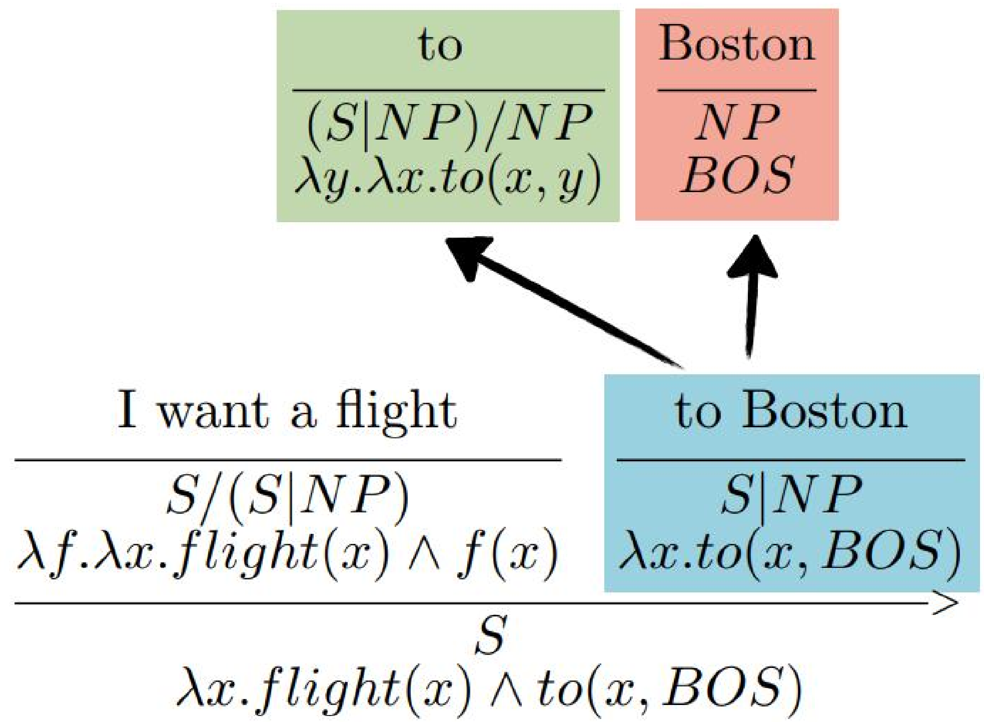
Learning Algorithm

I want a flight to Boston

$\lambda x.flight(x) \land to(x, BOS)$

1. Find highest scoring correct parse
2. Find splits that most increases score
3. Return new lexical entries

Iteration 2



Learning Algorithm

I want a flight to Boston

$\lambda x.flight(x) \wedge to(x, BOS)$

1. Find highest scoring correct parse
2. Find splits that most increases score
3. Return new lexical entries

Iteration 2

$$\dfrac{\text{to}}{\begin{array}{c}(S|NP)/NP \\ \lambda y.\lambda x.to(x, y)\end{array}} \qquad \dfrac{\text{Boston}}{\begin{array}{c}NP \\ BOS\end{array}}$$

$$\dfrac{\dfrac{\text{I want a flight}}{\begin{array}{c}S/(S|NP) \\ \lambda f.\lambda x.flight(x) \wedge f(x)\end{array}} \quad \dfrac{\text{to Boston}}{\begin{array}{c}S|NP \\ \lambda x.to(x, BOS)\end{array}}}{\begin{array}{c}S \\ \lambda x.flight(x) \wedge to(x, BOS)\end{array}} >$$

Experiments

System	Variable Free			Lambda Calculus		
	Rec.	Pre.	F1	Rec.	Pre.	F1
Cross Validation Results						
KRISP	71.7	**93.3**	81.1	–	–	–
WASP	74.8	87.2	80.5	–	–	–
Lu08	81.5	89.3	**85.2**	–	–	–
λ-WASP	–	–	–	86.6	92.0	89.2
Independent Test Set						
ZC05	–	–	–	79.3	**96.3**	87.0
ZC07	–	–	–	86.1	91.6	88.8
UBL	81.4	89.4	**85.2**	85.0	94.1	**89.3**
UBL-s	**84.3**	85.2	84.7	**87.9**	88.5	88.2

(on Geo880)

Experiments

System	English			Spanish		
	Rec.	Pre.	F1	Rec.	Pre.	F1
λ-WASP	75.6	91.8	82.9	80.0	92.5	**85.8**
UBL	78.0	**93.2**	**84.7**	75.9	**93.4**	83.6
UBL-s	**81.8**	83.5	82.6	**81.4**	83.4	82.4
System	**Japanese**			**Turkish**		
	Rec.	Pre.	F1	Rec.	Pre.	F1
λ-WASP	81.2	90.1	**85.8**	68.8	90.4	**78.1**
UBL	78.9	**90.9**	84.4	67.4	**93.4**	**78.1**
UBL-s	**83.0**	83.2	83.1	**71.8**	77.8	74.6

(on Geo250)