TokensRegex

August 15, 2013

Angel X. Chang

TokensRegex

- Regular expressions over tokens
- Library for matching patterns over tokens
- Integration with Stanford CoreNLP pipeline
  - access to all annotations
- Support for multiple regular expressions
  - cascade of regular expressions (FASTUS-like)

- Used to implement SUTime

- http://nlp.stanford.edu/software/tokensregex.shtml

Motivation

- Complementary to supervised statistical models
  - Supervised system requires training data
  - Example: Extending NER for shoe brands
- Why regular expressions over tokens?
  - Allow for matching attributes on tokens (POS tags, lemmas, NER tags)
  - More natural to express regular patterns over words (than one huge regular expression)

Annotators

- TokensRegexNERAnnotator
  - Simple, rule-based NER over token sequences using regular expressions
  - Similar to RegexNERAnnotator but with support for regular expressions over tokens
- TokensRegexAnnotator
  - More generic annotator, uses TokensRegex rules to define patterns to match and what to annotate
  - Not restricted to NER

TokensRegexNERAnnotator

- Custom named entity recognition
- Uses same input file format as RegexNERAnnotator
  - Tab delimited file of regular expressions and NER type
  - Tokens separated by space
  - Can have optional priority
- Examples:

```
San Francisco   CITY
Lt\. Cmdr\.     TITLE
<?[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}>?     EMAIL
```

- Supports TokensRegex regular expressions for matching attributes other than text of token

```
( /University/ /of/ [{ ner:LOCATION }] )       SCHOOL
```

TokensRegex Patterns

- Similar to standard Java regular expressions
- Supports wildcards, capturing groups etc.
- Main difference is syntax for matching tokens

Token Syntax

- Token represented by `[ <attributes> ]`
  `<attributes> = <basic_attrexpr> | <compound_attrexpr>`
- Basic attribute
  - form `{ <attr1>; <attr2> … }`
  - each `<attr>` consist of `<name> <matchfunc> <value>`
- Attributes use standard names (word, tag, lemma, ner)

Token Syntax

Attribute matching

- String Equality: `<attr>:"text"`

  `[ { word:"cat" } ]` matches token with text "cat"

- Pattern Matching: `<name>:/regex/`

  `[ { word:/cat|dog/ } ]` matches token with text "cat" or "dog"

- Numeric comparison: `<attr> [==|>|<|>=|<=] <value>`

  `[ { word>=4 } ]` matches token with text of numeric value >=4

- Boolean functions: `<attr>::<func>`

  `word::IS_NUM` matches token with text parsable as number

Token Syntax

Compound Expressions: compose using `!`, `&`, and `|`

- Negation: `!{X}`

  `[ !{ tag:/VB.*/ } ]` any token that is not a verb

- Conjunction: `{X} & {Y}`

  `[ {word>=1000} & {word <=2000} ]`

  word is a number between 1000 and 2000

- Disjunction: `{X} | {Y}`

  `[ {word::IS_NUM} | {tag:CD} ]` word is numeric or tagged as CD

- Use `()` to group expressions

Sequence Syntax

Putting tokens together into sequences

- Match expressions like “from 8:00 to 10:00”

```
/from/ /\\d\\d?:\\d\\d/ /to/ /\\d\\d?:\\d\\d/
```

- Match expressions like “yesterday” or “the day after tomorrow”

```
(?: [ { tag:DT } ] /day/ /before|after/)?
/yesterday|today|tomorrow/
```

- Supports wildcards, capturing / non-capturing groups and quantifiers

Using TokensRegex in Java

TokensRegex usage is like `java.util.regex`

- Compile pattern

```
TokenSequencePattern pattern =
      TokenSequencePattern.compile("/the/ /first/ /day/");
```

- Get matcher

```
TokenSequenceMatcher matcher = pattern.getMatcher(tokens);
```

- Perform match

```
matcher.matches()
matcher.find()
```

- Get captured groups

```
String matched = matcher.group();
List<CoreLabel> matchedNodes = matcher.groupNodes();
```

Matching Multiple Regular Expressions

- Utility class to match multiple expressions

```
List<CoreLabel> tokens = ...;

List<TokenSequencePattern> tokenSequencePatterns = ...;

MultiPatternMatcher multiMatcher =
   TokenSequencePattern.getMultiPatternMatcher(
     tokenSequencePatterns
   );

List<SequenceMatchResult<CoreMap>>
     multiMatcher.findNonOverlapping(tokens);
```

- Define rules for more complicated regular expression matches and extraction

Extraction using TokensRegex rules

- Define TokensRegex rules
- Create extractor to apply rules

```
CoreMapExpressionExtractor extractor =
CoreMapExpressionExtractor.createExtractorFromFiles(
   TokenSequencePattern.getNewEnv(), rulefile1, rulefile2,...);
```

- Apply rules to get matched expression

```
for (CoreMap sentence:sentences) {
   List<MatchedExpression> matched =
     extractor.extractExpressions(sentence);
   ...
}
```

- Each matched expression contains the text matched, the list of tokens, offsets, and an associated value

TokensRegex Extraction Rules

- Specified using JSON-like format
- Properties include: rule type, pattern to match, priority, action and resulting value
- Example

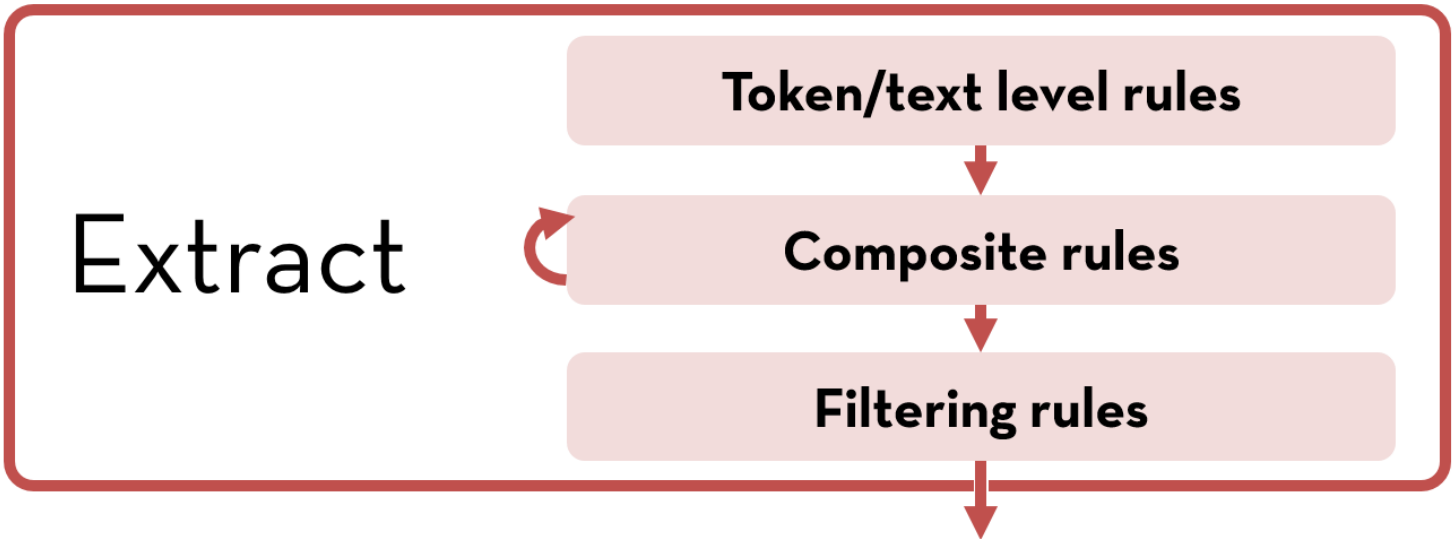
```
{
ruleType: "tokens“,

pattern: (([{ner:PERSON}]) /was/ /born/ /on/ ([{ner:DATE}])),

result: "DATE_OF_BIRTH“
}
```

TokensRegex Rules

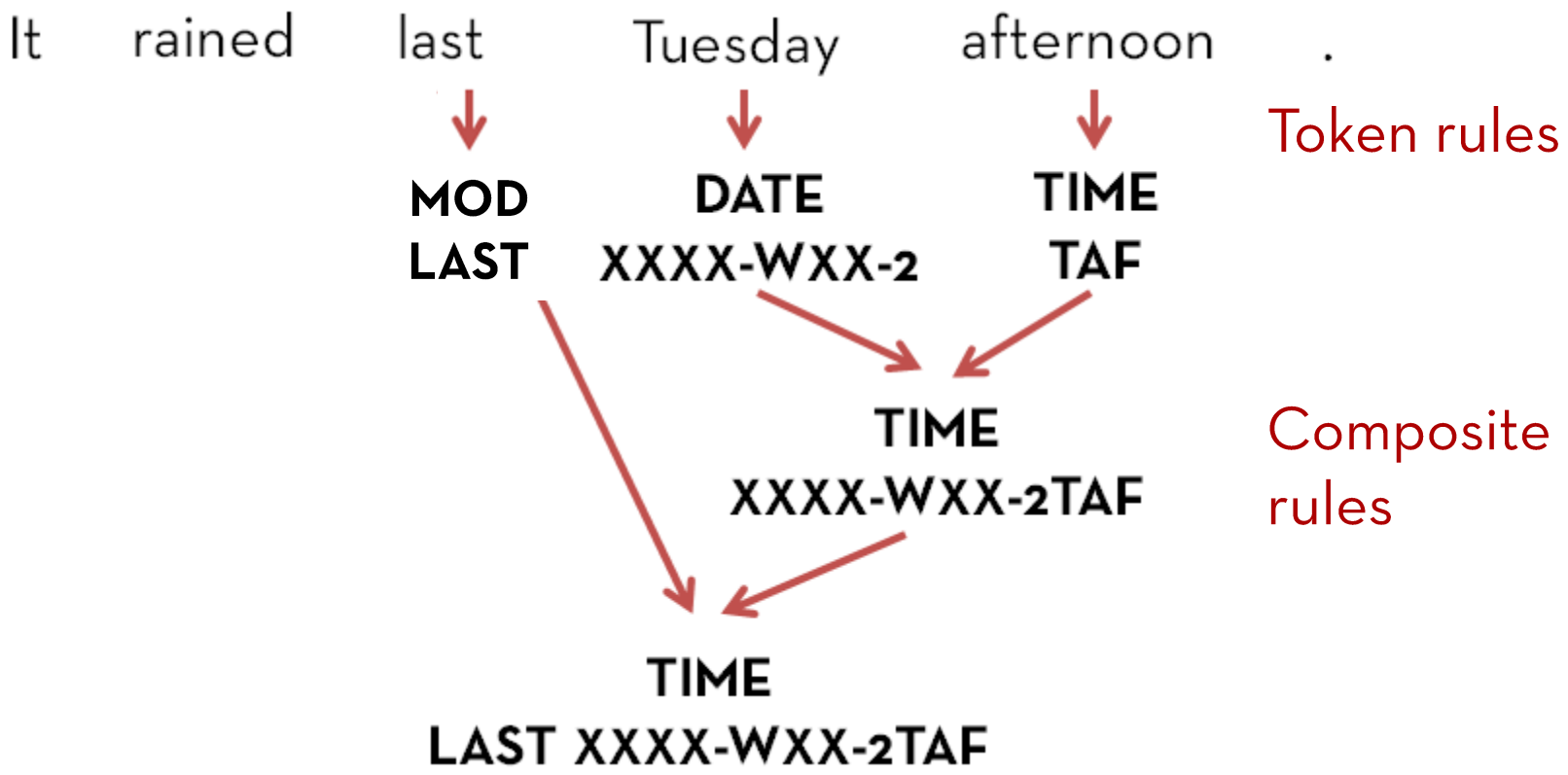
- Four types of rules
  - **Text:** applied on raw text, match against regular expressions over strings
  - **Tokens:** applied on the tokens and match against regular expressions over tokens
  - **Composite:** applied on previously matched expressions (text, tokens, or previous composite rules), and repeatedly applied until no new matches
  - **Filter:** applied on previously matched expressions, matches are filtered out and not returned

TokensRegex Extraction Pipeline

- Rules are grouped into stages in the extraction pipeline
- In each stage, the rules are applied as in the diagram below:



SUTime Example



TokensRegexAnnotator

- Fully customizable with rules read from file
- Can specify patterns to match and fields to annotate

```
# Create OR pattern of regular expression over tokens to hex RGB
code for colors and save it in a variable
$Colors = (
  /red/     => "#FF0000" | /green/   => "#00FF00" |
  /blue/    => "#0000FF" | /black/   => "#000000" |
  /white/   => "#FFFFFF" | (/pale|light/) /blue/   => "#ADD8E6"
)
# Define rule that upon matching pattern defined by $Color
annotate matched tokens ($0) with ner="COLOR" and
normalized=matched value ($$0.value)
{ ruleType: "tokens",
  pattern: ( $Colors ),
  action:  ( Annotate($0, ner, "COLOR"), Annotate($0, normalized,
           $$0.value ) ) }
```

The End

- Many more features!
- Check it out: http://nlp.stanford.edu/software/tokensregex.shtml