Масштабируемая среда для многоуровневого анализа текстов

Носков Алексей

МГУ им. М.В.Ломоносова, ВМК

alexey.noskov@gmail.com

20 октября 2011

Обработка ЕЯ-текстов

Задачи, связанные с анализом текстовой информации на естественном языке

- Извлечение информации из текстов
- Автоматическое реферирование
- Автоматическая разметка текстов
- Классификация текстов
- Поиск в текстовых корпусах

Построение ЕЯ-приложений

Построение приложений обработки ЕЯ-текстов предполагает интеграцию различных компонентов, реализующих сложные алгоритмы на различных уровнях.

При обработке больших объемов данных возникают проблемы масштабирования приложения.

Метод борьбы со сложностью - переиспользование кода.

Необходимы инструментальные средства разработки ЕЯ-приложений, обеспечивающие переиспользование кода.

Инструментальные средства разработки ЕЯ-приложений

- Позволяют создавать приложения, осуществляющие обработку ЕЯ-текстов
- Предоставляют готовые элементы для приложений:
  - Концепции (форматы данных, описания протоколов, архитектурные решения)
  - Программный код (осуществляющий работу с форматами, решающий лингвистические задачи, обеспечивающий взаимодействие других элементов)
- Должны решать основные проблемы разработки ЕЯ-приложений

Проблемы разработки ЕЯ-приложений

- Совместимость компонентов и структур данных
- Взаимозаменяемость компонентов
- Конфигурация приложения и компонентов
  - Настройка параметров компонентов
  - Указание источников данных
- Поддержка вспомогательных задач приложения
  - Оценка качества решения задач
  - Обучение компонентов машинного обучения
- Масштабирование приложения
  - Организация параллельных и распределенных вычислений

Содержание

Инструментальная среда LT-NSL

- ***Компоненты*** (стеммеры, графематические анализаторы) реализуются в виде отдельных процессов ОС
- Компоненты взаимодействуют через стандартные ввод и вывод (соединяются через UNIX-каналы)
- В памяти не хранится весь текст - только необходимая для обработки часть

- Нет контроля совместимости и взаимозаменяемости компонентов
- Вспомогательные задачи отдельно от приложения
- Масштабирование вручную средствами ОС

D. McKelvie, C. Brew, and H. Thompson, "Using SGML as a Basis for Data-Intensive NLP"IN PROCEEDINGS OF THE FIFTH CONFERENCE ON APPLIED NATURAL LANGUAGE PROCESSING (ANLP-97, 1997)

Инструментальная среда GATE

- Общая форма для обмена информацией в форме аннотаций
- Библиотеки для разработки и тестирования компонентов
- Графическая среда для построения ЕЯ-приложений

- Нет контроля совместимости и взаимозаменяемости компонентов
- Масштабирование вручную вне среды

H. Cunningham, K. Humphreys, and R. Gaizauskas, "GATE - a TIPSTER-based General Architecture for Text Engineering,"IN PROCEEDINGS OF THE TIPSTER TEXT PROGRAM (PHASE III) 6 MONTH WORKSHOP. DARPA, 1997.

Библиотека OpenNLP

- Различные компоненты: разбиение текста на слова и предложения, определение частей речи, синтаксический анализ...
- Для каждого типа компонентов есть интерфейс, контролирующий совместимость

- Нет разделения на код приложения и его конфигурацию
- Реализация вспомогательных задач вручную, как отдельных приложений
- Масштабирование вручную непосредственно в коде приложения

http://opennlp.sourceforge.net/

Содержание

Совместимость компонентов и структур данных

Виды несовместимости

- Несовместимость представлений – компоненты используют одинаковую модель данных, но различные представления данных (структуры, обозначения)
  - Решается преобразованиями структур данных в рамках модели
  - Предупреждается за счет использования стандартного набора структур данных в рамках модели
- Несовместимость моделей – компоненты используют различные модели данных
  - Решается преобразованиями структур данных между моделями

Совместимость компонентов и структур данных

Модели данных среды

- ***Структурная модель***. Текст – последовательность предложений, предложение – последовательность слов + набор связей
- ***Аннотационная модель***. Текст – строка символов + набор аннотаций
- ***Признаковая модель***. Текст – набор признаков

Один компонент как правило работает в рамках одной модели и совместим только с компонентами той же модели

Совместимость компонентов и структур данных

Направления решения

- Открытость для использования различных моделей обработки текстовых данных
- Максимальное переиспользование сущностей между стандартными моделями
- Наличие средств преобразования данных между моделями
- Наличие средств модификации компонентов для использования в другой модели

Совместимость компонентов и структур данных

Переиспользование элементов модели данных

Слово в структурной модели:

```
case class Word(normal: String, features: Map[String,Any])
```

Аннотация:

```
case class Annotation[T](start: Int, end: Int, data: T)
```

Аннотирование фрагмента как слова:

```
Annotation(1, 5, Word("run", Word.TENSE -> Time.PAST))
```

Совместимость компонентов и структур данных

Модификация компонентов

Морфологический анализатор, как компонент структурной модели:

```
trait Morphology extends Component {

  def analyze(s: String): Alt[Word]

}
```

Его модификация для аннотационной модели:

```
class MorphologyExtension(morph: Morphology) {

  def analyze(as: List[Annotation[Any]]):
    List[Annotation[Word]] = ...

}
```

Взаимозаменяемость компонентов

- Разделение интерфейса и реализации
- Использование в логике приложения интерфейсов
- Реализация компонентов - элемент конфигурации
- Использование параметризации

```
trait AnnotationProcessor[F,T] extends Component {
  def process(l: List[Annot[F]]): List[Annot[T]]
}
```

Конфигурация приложения и компонентов

- Какие реализации компонентов использовать
- Какие параметры им передавать
- Как менять поведение компонентов с помощью декораторов

Специальный DSL (Domain Specific Language) для конфигурации

```
component[Morphology] { comp =>
  comp implementedWith classOf[LuceneMorphology]
  comp hasOption ("language" -> "russian")
  comp decoratedWith ThreadPoolDecorator(3)
}
```

Поддержка вспомогательных задач приложения

- Задачи уровня приложения
- Задачи компонентов
- Зависимости между задачами

DSL для определения задач непосредственно в коде приложения

```
command("prepare-data") { env =>
  ...
}
```

Выполнение задач при старте приложения

```
> app start
> app morphology console
```

Масштабирование приложения

Проблемы

- Необходимость распараллеивания и реализации распределенности
- Код управления вычислениями переплетается с кодом решения задачи

Необходимо отделить код решения задачи от механизма вычислений

Масштабирование приложения

Абстрактные вычисления

Средство, позволяющее предствалять абстрактные вычисления - ***монады***

Две операции:

- ***map*** (>>) - применение некоторой функции к результату вычисления
- ***bind*** (>>=) - создание нового вычисления на основе результата

Виды вычислений:

- Непосредственное
- Отложенное
- Удаленное

Eugenio Moggi. "Notions of computation and monads"Inf. Comput. 93, 1 (July 1991), 55-92

Масштабирование приложения

Компоненты

Компоненты абстрагированы от модели вычислений

- Каждая функция компонента возвращает не результат, а вычисление результата
- Специальные объекты - ***декораторы*** модифицируют поведение компонентов
- Декораторы могут менять модель вычисления компонента
  - Выполнять функции компонента в пуле потоков
  - Выполнять функции компонента на другом вычислительном узле

Масштабирование приложения

Пример

Разбиение текста на лексемы и морфологический анализ всех слов:

```
tokenizer.tokenize(text) >>>= morphology.analyze
```

Модификаиця компонента морфологического анализа для выполнения в пуле потоков:

```
comp decoratedWith ThreadPoolDecorator(5)
```

Текущее состояние

Реализованные компоненты

- Стеммеры на базе Snowball
- Морфологические анализаторы на базе Mystem и Lucene
- Компоненты выделения слов и предложений на базе регулярных выражений
- Компоненты на основе библиотеки OpenNLP
- Компонент выделения аннотаций на основе регулярных преобразователей

Текущее состояние

Реализованные приложения

Выделение информации о спортивных событий из новостных текстов

- Определение видов спорта (футбол, хоккей, воллейбол...)
- Выделение упоминаний кубков (кубок Стенли, кубок мира)
- Выделение упоминаний матчей (ЦСКА сыграло со Спартаком со счетом 2:0)
- 150 строк на Scala

Результаты работы

- Реализована инструментальная среда разработки ЕЯ-приложений
  - Осуществляющая контроль совместимости и взаимозаменямости компонентов
  - Поддерживающая конфигурацию приложения отдельно от кода решаемой задачи
  - Предоставляющая средства для задания вспомогательных задач приложения
  - Предоставляющая средства для масштабирования приложений без изменения кода решения задачи

Дальнейшие направления развития

- Реализация дополнительных компонентов
- Реализация тестовых приложений
- Утилиты для упрощения создания приложений
- Графический интерфейс для создания простейших приложений
- Средства реализации абстрактного интерфейса пользователя на основе абстрактных вычислений

Вопросы?

Язык реализации - Scala

Scala - относительно новый язык для платформы Java, разрабатываемый с 2001 года в EPFL

- Поддерживает функциональное программирование
- Осуществляет автоматический вывод типов
- Легко расширяем

http://www.scala-lang.org/

Компоненты

- Интерфейсы – Scala Traits (аналог интерфейсов Java)
- Реализации – обычные классы, создаются с помощью фабричных функций
- Расширения – неявные преобразования
- Декораторы – классы, осуществляющие преобразование фабричных функций

Компоненты: языки реализации

- ***Scala***
- ***Java*** – в силу совместимости со Scala
- ***Python***, ***JavaScript***, ***Ruby*** – с использованием реализаций этих языков для JVM
- ***C / C++*** – с использованием механизма JNI

Связывание вычислений в приложении

- `>>` – сокращение для ***map***
- `>>=` – сокращение для ***bind***
- Использование записи на основе ***for***

```
markerAnnotator.annotate(text) >>=
  matchAnnotator.annotateReplace >>>=
  mapMatch(entry)

for (
  team1 <- teamName(ann.data("team1"))
  team2 <- teamName(ann.data("team2"))
) yield Match(team1, team2)
```

Компоненты: язык регулярных преобразователей

```
n("Vs") | n("Win").$("rf") | n("Loose").$("rf")
```

```
text("cup") ~ nounPhrase(Case.GENITIVE) ==>
    g("Cupb")
```

```
n("Team").$(1) ~ res.opt ~ n("Team").$(2) ==>
    g("Match", "team1" -> 1, "team2" -> 2)
```