А. С. Одинокова, М. С. Кудинов
Московский государственный университет им. М. В.Ломоносова
(Россия, Москва)

ПРОСТОЙ ПОДХОД К АВТОМАТИЧЕСКОМУ КОЛИЧЕСТВЕННОМУ АНАЛИЗУ ПОЭТИЧЕСКОГО ТЕКСТА НА ПРИМЕРЕ РОМАНА В СТИХАХ «ЕВГЕНИЙ ОНЕГИН»

Методы обработки естественного языка широко используются для решения прикладных (веб-поиск, классификация документов), так и научных проблем (исследование семантики слов с использованием моделей word2vec). Тем не менее, для анализа поэзии такие методы используются не так часто. В этой статье мы демонстрируем простой пример подхода к задаче анализа данных, применительно к поэтическим текстам. Мы показываем, что простой набор утилит для обработки текста может сэкономить значительное количество ручной работы. Набор состоит из бесплатного морфологического анализатора Yandex mystem, простого веб-приложения для ручного снятия неоднозначности, реализованного на JavaScript, и утилит на Python для определения ударения в словах и агрегации данных. Обработка текста опиралась на регулярные выражения. Подход был применен к классическому тексту «Евгений Онегин». Текущий уровень развития современных языков программирования и соответствующих библиотек (например, библиотека nltk для Python) позволяет еще быстрее разрабатывать приложения для анализа данных для текстов. Тем не менее, подход, описанный в статье, может быть использован в качестве примера для дальнейшего количественного исследования поэтических текстов.

Ключевые слова: автоматический анализ стиха, ритмика прилагательных

Введение

Автоматическая обработка текста уже давно превратилась в самостоятельную прикладную дисциплину, имеющую широкое применение как на потребительском рынке (например, в поисковых технологиях и кластеризации новостей), так и в создании инструментов исследования для других наук. Последнее применение можно хорошо проиллюстрировать на примере Национального корпуса русского языка. Тем не менее, данные методы используются не достаточно широко, когда речь идет об исследовании поэтических текстов. Между тем, использование подобных

методов может серьезно облегчить задачу исследователя, а разработка программы, автоматизирующей анализ, может при должном подходе быть произведена в сжатые сроки. Ниже будет приведен пример программы, помогающей существенным образом облегчить процесс анализа поэтического текста большого объема, — анализу будет подвергнут текст романа в стихах «Евгений Онегин».

Стоит отметить, что разработка описываемого ниже метода изначально не представляла собой самостоятельную и самодостаточную задачу: потребность в данном решении возникла в результате необходимости обработки большого текстового массива в весьма сжатые сроки, причем разработка велась по мере возникновения новых задач. Таким образом, стоит иметь в виду, что не все частные решения в рамках данного метода являются безупречными с точки зрения удобства использования и общего дизайна. Некоторые замечания и возможные улучшения также будут оговорены ниже.

Описываемая ниже программа была реализована на Python.

Постановка задачи

Исходной целью исследования было получение статистических таблиц, отражающих зависимость ритмической структуры прилагательных от их морфологических характеристик — рода, числа, падежа, степени сравнения и полной или краткой формы. В общем виде таблицы для некоторой грамматической категории *Cat* должны были выглядеть так:

Кат\структура	**1/1**	**2/2**	**3/3**	**…**	**m/k**
Cat1					
Cat2					
…					
CatN					

где k — ударный слог в слове, m — количество слогов.
m/k — характеризует собой тип структуры — количество слогов и ударный слог. В каждой ячейке содержится количество слов данной структуры, имеющих данное значение категории *Cat*.

Изначально все подсчеты предполагалось проводить вручную, однако предполагаемый объем работы был слишком внушительным: так, текст «Евгения Онегина» содержит более пяти тысяч строк, из которых немногим более двух тысяч содержат прилагательные. Грубый подсчет (после часовой работы) показал, что на обработку строк с прилагательными потребовалось бы не менее 30 человекочасов, при условии, что производительность остается постоянной (что, вообще говоря, не верно). Дополнительного времени потребовал бы также просмотр строк, не содержащих прилагательные. При этом процесс анализа можно легко автоматизировать, воспользовавшись свободно распространяемыми программными продуктами — например, морфологическим анализатором.

Очевидно, однако, и то, что полной автоматизации добиться не возможно по причине омонимии. Это автоматически означает, что программа должна

предусматривать возможность интерактивной или пост-обработки. То же можно сказать и о процедуре акцентуации.

На основании рассуждений, приведенных выше, сформулируем следующие этапы обработки:

1. Выделение прилагательного в тексте.
2. Определение грамматических характеристик прилагательного в автоматическом режиме.
3. Пост-обработка (в случае невозможности реализации интерактивного метода).
4. Определение места ударения в прилагательных.
5. Пост-обработка (в случае невозможности реализации интерактивного метода).
6. Подсчет количества слогов в прилагательном.
7. Составление таблиц.

Ниже рассмотрим каждый из перечисленных этапов подробнее.

Морфологический анализ

Для реализации первых двух этапов обработки было решено использовать программу Yandex.Mystem — свободно распространяемый морфологический стеммер компании «Яндекс» (адрес http://company.yandex.ru/technologies/mystem). В узком смысле стеммером называется алгоритм, позволяющий выделить основу для некоторого входного слова *w.* В более широком смысле стеммером называется морфологический анализатор.

Для демонстрации возможностей программы приведем пример со страницы программы.

Например, при подаче на вход Yandex mystem текста «В мурелки шлепают пельсиски» на выходе программа выдает следующий текст:

```
$ mystem -nig sapgir.txt
В{в=PR=|в=S,сокр=(им,ед|им,мн|род,ед|род,мн|дат,ед|дат,мн|вин,ед|вин,мн|твор,ед
|твор,мн|пр,ед|пр,мн)}
мурелки{мурелка?=S,жен,од=(им,мн|род,ед)|мурелка?=S,жен,неод=(им,мн|
род,ед|вин,мн)|мурелки?=S,мн,неод=(им|вин)|мурелок?=S,муж,од=им,мн}
шлепают{шлепать=V,несов=непрош,мн,изъяв,3-л,пе}
пельсиски{пельсиска?=S,имя,ед,жен,од=род|пельсиска?=S,жен,неод=
(им,мн|род,ед|вин,мн)}
```

В фигурных скобках после каждой словоформы указывается грамматическая информация. Подробное описание грамматических помет можно найти на странице программы. В основном грамматические пометы являются достаточно понятными даже для человека знакомого с русистикой только в рамках школьной программы. Обратим внимание лишь на то, что часть речи обозначается латинской буквой в соответствии с латинским наименованием. Для интересующих нас прилагательных это будет соответственно A.

Этот пример демонстрирует способность программы при наличии неизвестного слова предложить варианты разбора, что весьма полезно при обработке фраз, отсутствующих в словаре mystem, например «*зарею поздной*».

Mystem предоставляет ряд опций командной строки. Например, команда

```
$ mystem -ci -e utf8 onegin.txt onegin_stemmed.txt
```

задает формат выдачи с печатью грамматической информации и с сохранением исходной словоформы. Результат сохраняется в файле onegin_stemmed.txt.

Первая строка романа в результате примет следующий вид:

```
Не{не=PART=} мысля{мыслить=V,несов=непрош,деепр,пе} гор-
дый{гордый=A=им,ед,полн,муж|=A=вин,ед,полн,муж,неод}
свет{свет=S,имя,муж,од=им,ед|света=S,имя,жен,од=род,мн|=S,имя,
жен,од=вин,мн|свет=S,ед,муж,неод=им|=S,ед,муж,неод=вин} заба-
вить{забавить=V,несов=инф,пе}
```

Очевидно, что редактирование текста в таком виде затруднено. Однако наличие отделенных от текста грамматических помет дает нам 2 возможности для упрощения работы разметчика:

1) В тексте можно оставить только те строчки, которые содержат прилагательные. С учетом статистики приведенной в п.1 количество просматриваемых строк сократится почти вполовину.

2) Все словоформы, не содержащие вариантов разбора в качестве прилагательных, могут быть выведены в выходной файл без грамматической информации.

Ориентируясь на частеречный показатель — в нашем случае A — программа-обработчик может автоматически отфильтровать строки не содержащие потенциальных прилагательных, при этом оставляя грамматическую информацию только у слов-претендентов.

В идеале мы ожидаем увидеть первый строку романа в следующем виде:

Не мысля гордый{гордый=A=им,ед,полн,муж|=A=вин,ед,полн,муж,неод} свет забавить,

Далее разметчик устраняет омонимию вручную.

Для достижения подобного результата удобнее всего воспользоваться аппаратом *регулярных выражений*. Неформально говоря, регулярные выражения представляют собой особую форму представления цепочек символов. Регулярное выражение выступает как форма записи строки образца, причем в составе образца могут быть метасимволы. К примеру, следующее регулярное выражение —

```
здра(?:сте|вствуйте) —
```

задает множество строк, среди которых содержатся *здрасте* и *здравствуйте*.

Регулярные выражения стали стандартом де-факто в обработке текста. Многие современные языки программирования (Perl, Python) имеют встроенную поддержку регулярных выражений. Регулярные выражения используются некоторыми текстовыми редакторами (MS Word, OpenOffice Writer) и командными интерпретаторами операционных систем (Unix shell) для поиска и подстановки текста.

Не вдаваясь в подробности синтаксиса регулярных выражений, разберем регулярное выражение, которое позволяет найти (распознать) прилагательное в выдаче mystem:

```
(\w+){((?:\w|\||,|=|\?)+?=A(=|,)(?:\w|\||,|=)+?)}                (1)
```

Часть (\w+) распознает цепочку буквенных символов, которая отождествляется со словоформой перед фигурными скобками.

(?:\w|\||,|=|\?)+? и (=|,)(?:\w|\||,|=)+?) — задают наборы символов, включаемых в грамматические пометы mystem. Например символы «,», «|», «=» и буквенные символы. Скобки обозначают группировку, а оператор +? указывает на то, что таких помет может быть не менее одной. Наконец, =A — распознает собственно подстроку «=A», которую всегда будет содержать грамматический разбор прилагательного.

Таким образом, регулярное выражение (1) находит все слова, в списке разборов которых есть частеречная помета A.

Несмотря на кажущуюся сложность, работа с регулярными выражениями на практике оказывается очень простой и эффективной, и не вызывает большого труда даже у начинающего программиста.

Таким образом, оказываются реализованы первые два этапа обработки.

Постобработка

Ручная постобработка в текстовом редакторе может по-прежнему оказаться долгой. В идеале постобработка должна выполняться в один клик. Обычно для реализации подобной функциональности пишутся небольшие оконные приложения. Разметка в таком случае, как правило, реализуется на XML. Однако те же возможности можно реализовать, написав буквально 50 строчках кода. Простой подход заключается в использовании стандартной для веб-программирования связки HTML+CSS+JavaScript. Рассмотрим это решение подробнее.

Структура документа HTML задает набор объектов документа (Document object model — DOM). Модель DOM не налагает ограничений на структуру документа. Любой документ известной структуры с помощью DOM может быть представлен в виде дерева узлов, каждый узел которого представляет собой элемент, атрибут, текстовый, графический или любой другой объект. Узлы связаны между собой отношениями «родительский — дочерний».

Так, например, для фрагмента разметки

```
<div id="id1"><span><a href="/">Ссылка</a></span></div>
```

верно, что ссылка <a> является потомком блока span, который лежит в блоке с идентификатором id1.

Браузеры опираются на свой движок, когда происходит преобразование (парсинг) HTML-файлов в DOM. Некоторые браузерные движки, к примеру Trident/MSHTML и Presto, тем или иным образом имеют привязку к определённому браузеру — Internet Explorer и Opera, соответственно. Такие же, как WebKit и Gecko, используются во множестве различных браузеров, таких как Safari, Google Chrome, Firefox или Flock.

Стандартный JavaScript, встроенный в браузер, позволяет манипулировать элементами DOM через инструкцию getElementById и дополнительные инструкции для передвижения по дереву.

```
obj = document.getElementById("navigation") //получить указа-
тель на объект с id="navigation"
```

Для облегчения работы с DOM существуют библиотеки, расширяющие функциональность стандартного JavaScript.

В частности, они могут предоставлять функции, облегчающие выборку элементов из структуры документа по какому-либо критерию, и передвижение по дереву. Одним из стандартов для организации навигации по документам является XPath. Xpath представляет собой язык запросов к элементам XML-документа. Поскольку HTML представляет собой подмножество XML, XPath может быть задействован для навигации по структуре веб-страницы. Оной из наиболее популярных на сегодняшний день библиотек, реализующих навигацию с помощью XPath-подобной нотации, является библиотека jQuery. Также jQuery позволяет обращаться к элементам с помощью селекторов CSS.

Синтаксис jQuery позволяет перемещаться по DOM и манипулировать объектами особенно просто:

```
$("div.test").addClass("blue"); //Всем элементам div с CSS-
классом test добавить оформление, описанное в CSS-классе blue
```

Теперь определим в файле hide.css классы так, что один из них отображается на веб-странице, а другой — нет:

```
/*не отображается на странице
.unchecked {
display: none;
}
/*отображается на странице

.anchecked {
display: inline;
}
```

Мы можем написать код на jQuery (см. Приложение 2) так, чтобы при нажатии на соответствующий элемент он получал класс anchecked, а все его братья в дереве — класс unchecked.

В другом файле correct.css эти же классы определяют просто цвета элементов:

```
.anchecked {
color: #f4af16;
}
.unchecked {
color: #cccccc;
}
```

Мы можем задать такое событие — например — нажатие на кнопку, которое добавит к описаниям классов в файле correct.css описания классов в файле hide.css. В результате все не выбранные разборы будут скрыты.

Для реализации данной идеи остается лишь соответствующим образом сгенерировать HTML-разметку и написать необходимый код на jQuery (соотв. код приведен в приложении).

Скрипт, обрабатывающий выдачу стеммера на предыдущем этапе, фильтрует ее и печатает результат в шаблон, содержащий базовые заголовки документа и ссылки на файлы .css и .js (таблицы стилей и код JavaScript соответственно). Строка может быть отформатирована, например, так:

```
<p>Не мысля <span class="target">гордый</span><span
class="angroup">{<span class="an">гордый=А=им,ед,полн,муж</
span><span class="an"><span>|</span>=А=вин,ед,полн,муж,неод</
span>}</span> свет забавить, </p>
```

Как видим все разборы слова имеют класс an, причем все они являются потомками элемента с классом angroup. Пользователь выбирает нужный разбор нажатием левой кнопки мыши. Данное событие обрабатывается следующим кодом:

```
$('.an').click(function() {
$(this).parent().children().attr('class', 'unchecked');
$(this).attr('class', 'anchecked');
});
```

Данный код присваивает всем потомкам родителя элемента, вызвавшего событие, класс unchecked, а самому элементу — класс anchecked.

Скрытие невыбранных элементов осуществляется по нажатию кнопки с id=”hide” кодом:

```
$('#hide').click(function() {
document.getElementById('hidecss').disabled = false;
});
```

Полностью автоматический анализ морфологии невозможен вследствие омонимии:

```
Музыка уж греметь устала{уставать=V,нп=прош,ед,изъ-
яв,жен,сов|усталый=А=ед,кр,жен}
Все было просто: пол
{пол=S,имя,муж,од=им,ед|пола=S,жен,неод=род,мн|пол=S,муж,не-
од=им,ед|= S,муж,неод=вин,ед|пол=NUM=им|=NUM=вин|по-
лый=А=ед,кр,муж} дубовый
```

Так как проблема автоматического снятия омонимии пока полностью не решена, морфологические характеристики прилагательных выбирались вручную. При снятии омонимии мы также столкнулись с проблемой субстантивации, то есть перехода прилагательного в существительное (например, больной, проездной, шашлычная), см. п.1.8. Спорные случаи трактовались в соответствии с «Грамматическим словарем русского языка» А. А. Зализняка для определения частеречной принадлежности.

Минусом подобного подхода является то, что обработанный материал трудно сохранить в файле: стандартный JavaScript не позволяет выполнять операции сохранения, поэтому самым удобным способом сохранения является копирование обработанного текста и вставка в текстовый документ.

Расстановка ударений

После постобработки морфологии можно перейти к работе с ударениями. Соображения относительно трудоемкости ручной обработки остаются справедливыми и в данном случае. Информация, которую необходимо извлечь на данном этапе — это количество слогов и место ударения в каждом из прилагательных.

Подсчет количества слогов в общем случае не вызывает затруднений: достаточно подсчитать количество гласных в слове. Это также делается при помощи регулярных выражений.

Определить место ударения можно двумя способами: 1) можно воспользоваться автоматическим акцентуатором, который произведет расстановку ударений на основе словаря; 2) воспользоваться тем фактом, что роман написан четырехстопным ямбом и ставить ударения на слоге в метрически сильной позиции.

У обоих методов есть недостатки. При использовании автоматического акцентуатора мы сталкиваемся с проблемой неполноты словаря: слова типа «полусмешных», «полупечальных» в нем отсутствуют; с другой стороны, при попытке расстановки ударений на каждом слоге в метрически сильной позиции обнаруживается, что не каждый слог на метрически сильной позиции является ударным, так как ритмические формы различаются расположением ударений в строке:

Залог достойнее тебя.

Полусмешных, полупечальных.

В качестве решения этой проблемы было выбрано совмещение двух вышеназванных методов с последующей ручной постобработкой тех случаев, когда метрическое и словарное ударения не совпадают.

Для обработки использовался автоматический акцентуатор А. Полякова (адрес http://feb-web.ru/febupd/parser/zip/accent.rar).

Обработка была организована следующим образом: вначале строки, полученные в результате постобработки, обрабатывались автоматическим акцентуатором. Результат сохранялся в выходном файле. Полученный файл обрабатывался Python-сценарием, расставляющим ударение на метрически сильной позиции. В случае несовпадения метрического и словарного ударений, строка сохранялась в файле для постобработки:

```
Прими собранье пёстрых<пестрый=A=род,мн,полн,ус=1,кс=2> глав,
```

В противном случае прилагательные с грамматической и акцентной информацией сохраняются в выходном файле:

```
гордый{=A=вин,ед,полн,муж,кс=2,ус=1}
```

В таком же виде представлялись данные в результате постобработки. Обработка не заняла много времени, поскольку в файле для постобработки оказалось чуть более трехсот строк на две тысячи триста семьдесят прилагательных.

Сбор статистики

Сбор статистики также был реализован на Python и задействовал регулярные выражения. Пример получившейся грамматической таблицы:

	1/1	2/2	3/3	4/4	2/1	3/2	4/3	5/4	3/1	4/2	5/3	6/4	4/1	5/2	6/3	7/4
ед	93	87	86	100	79	79	76	83	69	63	65	54	74	64	88	0
мн	7	11	14	0	20	19	24	15	31	35	35	46	26	36	13	100
n/a	0	3	0	0	1	2	1	2	0	2	0	0	0	0	0	0

Таблица показывает распределение ритмических структур по числу. Значение n/a выставлялось для словоформ, грамматические показатели которых блокируют выражение числа (например, сравнительная степень).

Вывод

Был продемонстрирован способ, позволяющий значительно сэкономить время на обработку большого текста и извлечение из него лингвистической информации. Разработка программы-обработчика при данном подходе занимает не более двух рабочих дней. Примерно столько же времени уходит на постобработку. Метод может быть легко адаптирован для других крупных поэтических и прозаических текстов.

Alexandra S. Odinokova, Mikhail S. Kudinov
Moscow State University
(Russia, Moscow)

A SIMPLE PIPELINE FOR QUANTITATIVE ANALYSIS OF «EUGENE ONEGIN»

Natural language processing methods are widely used for solving both applied (web search, document classification) and scientific problems (distributional semantics research with word2vec models). However, such methods are not widely used for analyzing poetry. In this paper we demonstrate a simple case of data analysis applied to poetic texts. We show that a simple pipeline of NLP utilities can save a huge amount of manual work. The pipeline consists of the free morphology analyzer Yandex mystem, a simple web application for manual morphological disambiguation implemented in JavaScript and Python utilities for word stress detection and data aggregation. Core text processing is based on regular expressions. The approach will be demonstrated using the classical text of *Eugene Onegin.* Note that the pipeline described in the paper was not a standalone application but an auxiliary utility developed for the purely linguistic task of analyzing the large text of *Eugene Onegin*. Thus there may be design flaws in the proposed approach. Moreover, the current level of development of modern programming languages and their corresponding libraries (e.g. nltk library for Python) allows for the even faster development of data analysis applications for texts. The pipeline described in the paper may be used as an example for further quantitative research of poetic texts.

*Key words***:** NLP, rhythmical structure of Russian adjectives.