МЕСТО ПУБЛИКАЦИИ: Бочаров В. В., Митренина О. В. Компьютерная морфология // Прикладная и компьютерная лингвистика / Николаев И.С., Митренина О.В., Ландо Т.М. (ред.), М.: URSS, 2017. С. 14–34.

Бочаров В. В., Митренина О. В.

Глава 1. Компьютерная морфология

1. Как найти слова

Морфология — это раздел лингвистики, который изучает структуру слов и их морфологические характеристики. Классическая морфология проанализирует слово *собака* примерно так: это существительное женского рода, оно состоит из корня *собак* и окончания *а*, окончание показывает, что слово употреблено в единственном числе и в именительном падеже.

Компьютерная морфология анализирует и синтезирует слова программными средствами. В наиболее привычной формулировке под морфологическим анализом подразумевается определение **леммы** (базовой, канонической формы слова) и его **грамматических характеристик**. В области автоматической обработки данных также используется термин «нормализация», обозначающий постановку слова или словосочетания в каноническую форму (грамматические характеристики исходной формы при этом не выдаются). Обратная задача, т.е. постановка леммы в нужную грамматическую форму, называется порождением словоформы.

Морфологическому анализу предшествует подготовительный этап: текст нужно разбить на предложения, а в каждом предложении выделить слова, знаки препинания и другие элементы текста — смайлики, числа, формулы, таблицы и пр. Этот этап называется **токенизацией** (tokenization), а выделенные в результате единицы (слова, числа, знаки препинания и пр.) называются **токенами** (tokens).

Задачи компьютерной морфологии, как и большинство задач прикладной лингвистики, решаются тремя типами подходов [Hajič 2004; Hajič et al. 2001; Коваль 2005; Леонтьева 2006]:

(1) подходы на основе правил, составленных экспертами (rule-based methods);

(2) статистические методы (statistical methods), связанные в основном с машинным обучением (machine learning);

(3) гибридные подходы, совмещающие статистику и правила (hybrid methods).

Токенизацию можно проводить с помощью любого из этих трех подходов.

Правила для выделения предложений на первый взгляд кажутся очевидными: предложения заканчиваются точкой, многоточием, вопросительным или восклицательным знаком. Можно рассматривать эти знаки как границы предложений. Но точкой могут заканчиваться также сокращения, например, *кв., гр., и пр.* и пр. Тогда можно добавить правило о том, что конец предложения отмечается точкой и следующей за ней прописной буквой. Под это правило ошибочно попадут инициалы, например, *Дж. Д. Сэлинджер,* и при этом не попадут предложения из социальных сетей, где прописные буквы используются не всегда. Однако, немного подумав, можно составить достаточно эффективный набор правил для выделения границ предложений.

Подходы, использующие статистические данные и методы машинного обучения, подробно описаны в пятой главе нашей книги. Они предполагают, что на вход компьютеру можно дать представительный корпус примеров с отмеченными границами предложений, и он по аналогии научится находить границы новых предложений, которых не было в этом корпусе.

Гибридный подход совмещает статистику и правила.

Слова (точнее, **словоформы** — слова в определенной грамматической форме) в русском языке выделять достаточно просто — у нас они разделяются пробелами, хотя и тут иногда возникают сложности. Гораздо сложнее дело обстоит в японском или в китайском языке — там пробелы используются далеко не всегда.

К счастью, большинство стандартных пакетов морфологического анализа включают в себя токенизацию, поэтому, скорее всего, вам не придется программировать эту процедуру с нуля.

2. Каким может быть анализ слов

Все выделенные в процессе токенизации словоформы надо проанализировать. Это значит, для каждой словоформы необходимо определить, (1) какие у нее грамматические признаки, (2) к какой части речи она относится и (3) в некоторых случаях (например, при машинном переводе, использующем словарь) надо понять, какой единице словаря соответствует данная словоформа.

Например, для первого слова *Решал* в предложении *Решал всё тот же я мучительный вопрос* необходимо (1) понять, что это глагол, (2) определить значения прошедшего времени, мужского рода и единственного числа, а затем, возможно, (3) сопоставить его с нужным разделом компьютерного словаря.

Поскольку память у современных компьютеров практически не ограничена, можно попытаться заложить в нее все словоформы и для каждой указать ее грамматические значения и другую необходимую информацию. Такой подход известен как **морфологический анализ со словарем словоформ**. В этом словаре будут целиком храниться, например, слова *решал, решает, решала, решаем* и др. Для каждого из них будет указано соответствующее ему время, число и род.

Такой словарь получится довольно большим, и значительная часть сведений в нем будет многократно повторяться. Он может применяться в тех языках, где слова мало изменяются грамматически. В них грамматическая информация передается не через изменение слов, а с помощью различных синтаксических средств: вспомогательных глаголов, предлогов, порядка слов и пр. Например, русский творительный падеж в слове *молотком* в английском языке передается с помощью предлога: *with a hammer*. Поэтому при обработке подобных языков основная грамматическая информация извлекается на следующем этапе — на уровне анализа предложения, которому посвящена следующая глава.

Главный недостаток словаря словоформ — в него невозможно включить абсолютно все слова живого языка. Ведь новые слова появляются очень часто, и человеческий мозг умеет их обрабатывать без словаря словоформ.

Глокая куздра штеко будланула бокра и курдячит бокрёнка — эта знаменитая фраза Л. В. Щербы показывает, что роль слова и его форму можно понять даже тогда, когда неизвестно его значение. Очевидно, что действия здесь осуществлялись *куздрой*, которая была *глокая*, и что действиям подверглись *бокр* и *бокренок*, эти действия назывались *будланут*ь и *курдячить,* первое действие уже завершилось, а второе продолжается. Мы можем понять это благодаря грамматике: формы слов и порядок их следования подсказывают, где в предложении глагол, а где существительные, и в каких грамматических формах они употреблены.

Технологии, позволяющие автоматически анализировать незнакомые слова, тоже существуют. Для их обработки применяется **предиктивный (предсказательный) морфологический анализ**, задача которого состоит в том, чтобы «угадать» грамматические характеристики слова и его основу, глядя на те его части, которые могли бы быть окончанием, суффиксами или приставками. Здесь годятся не только правила, но и машинное обучение: компьютер обучается на имеющемся словаре лемм и словоформ и сравнивает новое слово,

которого нет в словаре, с имеющимися, чтобы выбрать наиболее подходящий образец. Этот подход может порождать и курьезы, например слово «кровать» может быть разобрано таким анализатором как глагол, поскольку оно похоже на глагол «ковать».

Но далеко не для всех языков удобен подход на основе традиционного морфологического представления, состоящего из перечисления склонений и спряжений. Представим, что нам нужно проанализировать слова немецкого языка для целей информационного поиска. В немецком возможны слова типа *Donaudampfschifffahrtskapitän* (капитан рейса, выполняемого пароходом по Дунаю). Если не делить его, то документ не будет найден по запросам, содержащим слова *Dona* (Дунай), *Dampfschiff* (пароход), *Fahrt* (рейс) и *Kapitän* (капитан), хотя он, вполне возможно, был бы релевантным ответом по этим запросам. Более того, составные части могут свободно сочетаться друг с другом, т. е. возможны слова *Newaschifffahrtkapitaen* (капитан рейса, выполняемого пароходом по Неве) и *Donaudampfschifffahrtsmatros* (матрос рейса, выполняемого пароходом по Дунаю). Аналогичным образом не удастся найти документ, в котором встретилось слово Anfangsschwierigkeiten (трудности начального периода) по запросу Schwierigkeit (трудность), если не разделить исходное слово на два. Таким образом, решая задачи, связанные с информационным поиском применительно к немецкому языку, словом удобнее считать отдельные составные части сложного слова.

Еще сложнее дело обстоит в турецком языке. Словоизменение там осуществляется путём прибавления аффиксов, обозначающих грамматические значения. При этом явного ограничения на количество этих аффиксов не существует, что позволяет составлять очень длинные слова. Вот пример из английской Википедии:

Muvaffak — успешный
Muvaffak**iyet** — успех
Muvaffakiyet**siz** — неуспешный
Muvaffakiyetsiz**leş** — стать неуспешным
Muvaffakiyetsizleş**tir** — сделать неуспешным кого-либо
Muvaffakiyetsizleştiri**ci** — тот, кто делает неуспешным кого-либо
Muvaffakiyetsizleştirici**leş** — стать тем, кто делает кого-либо неуспешным

и далее можно продолжать добавлять морфемы справа, пока не получится слово *Muvaffakiyetsizleştiricileştiriveremeyebileceklerimizdenmişsinizcesine*. Его можно перевести примерно так: *Как будто бы ты оказался среди тех, кого нам непросто будет сделать теми, кто делает кого-либо неуспешным*.

Очевидно, что для целей информационного поиска или автоматического перевода нужно проанализировать структуру таких слов, отделяя каждую морфему.

Составить словарь всех возможных словоформ турецкого языка так, как, например, это делается для русского или английского, не представляется возможным. Один из вариантов решения задачи морфологического анализа для языков такого типа — это объединение словаря морфем и набора правил, задающих их возможные сочетания. Так устроен, например, свободно доступный морфологический анализатор для турецкого языка TRmorph.

3. Лексическая неоднозначность

Многие слова допускают неоднозначный анализ — в этом случае говорят о лексической или морфологической неоднозначности. Например, словоформа *стали* может быть образована как от существительного *сталь*, так и от глагола *стать*, а словоформа *море* может быть связанна с тремя существительными — *море* (часть Мирового океана), *мор* (эпидемия) и *мора* (единица в стихосложении).

Самые распространенные типы лексической неоднозначности — это **омонимия** и **полисемия**. При омонимии совпадают совершенно разные слова (например, *лук* может обозначать оружие или растение), а полисемия возникает там, где у одного слова

проявляется несколько разных значений. Известный пример полисемии — это слово *взять,* которое может обозначать до пятидесяти разных действий: принять в руку, получить в свое пользование, арестовать, принять на работу и пр. Этот вид неоднозначности важен при машинном переводе, когда нам нужно понять правильное значение слова.

Частичная лексическая омонимия — это совпадение отдельных форм слов, они называются омоформами. Слово *физики* может обозначать группу *ученых-физиков* или быть родительным падежом слова *физика.*

Грамматическая омонимия — совпадение форм одного слова. Например, у слова *дочь* совпадают формы именительного и родительного падежей: *Дочь обедает* и *Вижу дочь.*

Сложные случаи неоднозначности могут быть вызваны **конверсией. Так называется** переход слова из одной части речи в другую. В русском языке наиболее частым примером конверсии является переход из прилагательного в существительное, например, слово *слепой.*

Сложность для анализа могут представлять также **омографы —** одинаковые по написанию слова, имеющие разное звучание и значение. Например, это первое слово в сочетании *большая часть (большАя* или *бОльшая?).* В устной речи им соответствуют **омофоны —** одинаковые по звучанию слова, отличающиеся по написанию. Например, сочетание *ко злу* (или *козлу*?) в предложении *Можно ли быть равнодушным ко злу.* Но омофоны — это проблема распознавания речи, которой посвящена третья глава.

В живом общении люди редко замечают языковую неоднозначность. Это говорит о том, что в языке есть эффективные способы уменьшения уровня неоднозначности. Вот некоторые их них:

1. Использование знаний о мире. Например, в предложении *Съешь лук* человек не проинтерпретирует слово *лук* как оружие для стрельбы — мы не рассматриваем здесь маловероятный случай сказочной истории, в которой чудовищу советуют обезоружить средневекового лучника, проглотив его оружие.

2. Использование контекста. Легко понять, о каком луке идет речь, если у нас есть сочетание *лук со стрелами* или *картошка с луком*.

3. Предпочтение самых частых форм. В предложении *Мужик купил порося* вряд ли кому-то придет в голову рассматривать слово *порося* как деепричастие от глагола *поросить*, хотя оно вполне может оказаться деепричастием.

4. Использование интонации — увы, этот метод не подходит для обычных письменных текстов.

Реализовать эти и другие способы борьбы с неоднозначностью — это одна из важных задач лингвиста-практика. Разрешение морфологической омонимии рассматривают как отдельную задачу, т.к. для неё необходимы дополнительные данные, в то время как её результат нужен не всегда. Методы разрешения морфологической неоднозначности разнообразны: они могут быть основаны на контекстных алгоритмах, на выводимых из текста правилах или на вероятностных моделях (с управляемым и с неуправляемым обучением), или использовать гибридные методы [Brill 1992; Protopopova, Bocharov 2013; Karlsson 1995; Shmidt 1994]. Во многих случаях (например, для последующего синтаксического анализа) наличие нескольких гипотез, одна из которых с большой степенью достоверности (> 99.9%) является правильной, предпочтительнее однозначного ответа, ошибочного даже в 0.1% случаев. Достижимой в настоящий момент точность при автоматическом снятии морфологической неоднозначности описана в [Шаров и др. 2015].

4. Анализ морфологии на основе правил

4.1. Что хранить в словарях?

Для многих языков, например английского или русского, морфологический анализ и синтез удобно проводить с помощью заранее созданных словарей, в которых тем или иным способом задан список базовых форм слов и всех соответствующих им словоформ. Например, в словаре должна храниться лемма (базовая форма) глагола *решать* с указанием

ее основы *реша-* и со ссылкой на другую базу, где хранятся все варианты концовок для этого слова и указанием их грамматических форм:

- л — 3-е лицо, единственное число, мужской род, прошедшее время;
- ла — 3-е лицо, единственное число, женский род, прошедшее время;
- ю — 1-е лицо, единственное число, настоящее время

и т.д.

В школьной грамматике основой считается часть слова без окончания. В большинстве случаев она не меняется при грамматических изменениях самого слова — так ведет себя, например, основа *слон* в словоформах *слон, слону, слонами, слонов*. Но в некоторых словах основа может изменяться. Например, для словоформ *день, дню* и *дне* основами будут *ден-, дн-* и *дн-*, такое явление называется **чередованием.** Поэтому самый популярный на сегодня подход использует **псевдоосновы** (или **машинные основы**). Это неизменяемые начальные части слов. Для слова *день* такой неизменяемой частью будет *д-*. А оставшиеся концы — **псевдофлексии —** могут заноситься в дополнительные таблицы или храниться как-то иначе. Каждой псевдофлексии присваивается набор грамматических значений. Так, *-ень* будет обозначать единственное число и именительный падеж для упоминавшийся псевдоосновы *д-*.

Формы некоторых слов могут образовываться от разных корней. Например, у слова *ходить* есть форма *шел.* Это называется **супплетивизмом**. В таких случаях, если не придумать какое-нибудь другое решение, псевдооснова получается нулевая, и все слово превращается в псевдофлексию. В русском языке супплетивизм и чередования очень распространены, поэтому псевдоосновы часто получаются очень короткими, а псевдофлексии — слишком сложными.

Если в главном словаре анализатора хранятся леммы — основные формы слов с указанием основы, то такой подход называют лемматизацией (**lemmatization**). Ему противопоставляют стемминг (**stemming**) — подход, при котором в словаре хранятся только основы. Существует бесплатный инструмент для написания стеммеров — Snowball. Он разработан для 20 языков, в том числе и для русского, и может быть реализован на многих языках программирования. Но для русского языка стемминг работает гораздо хуже, чем лемматизация.

Современные модули морфологического анализа для распространённых языков используют словарные базы данных объемом не менее 100 тысяч базовых форм слов. Вся русская морфология чаще всего базируются на грамматическом словаре А.А. Зализняка [Зализняк 1977]. Он был впервые издан в 1977 году и с тех пор неоднократно переиздавался. В данный момент в сети доступны отсканированные страницы этого словаря и различные базы данных, построенные на его основе. Бумажный словарь содержит систематизированное описание 100 тысяч слов с указанием особенностей их словоизменения.

Чем больше компьютерный словарь, тем лучше, однако всегда можно найти слова, которых в словаре не окажется: это могут быть новые слова, узкоспециализированные термины, жаргонизмы, слова с опечатками. Для них необходима система предиктивного анализа.

Далее мы рассмотрим некоторые доступные модули для морфологического анализа русского языка. Они различаются составом и объёмом словаря, скоростью поиска по нему, набором тегов, обозначающих грамматические признаки, способом обработки несловарных слов и наличием дополнительных функций (например, ранжированием гипотез или снятием неоднозначности с учётом контекста). Кроме этого, различаются юридические условия их использования (лицензия, указывающая, в частности, разрешено ли использование в коммерческих целях), доступность исходного кода модуля, возможность пополнения словаря новыми словами, не привлекая к этому разработчиков. Существуют также разные варианты технического оформления модуля: отдельная программа, библиотека для языка C/C++, модуль для языка Python и т.д.

4.2. Морфологические модули АОТ

Рабочая группа АОТ (Автоматическая обработка текста, руководитель — А. В. Сокирко) выкладывает свои разработки в свободный доступ начиная с 2002 года. Их библиотеки можно использовать бесплатно даже в коммерческих проектах. Проект включает в себя модули для проведения графематического, морфологического, синтаксического и семантического анализа. Разработчики реализовали снятие морфологической неоднозначности с использованием скрытых марковских моделей и синтаксический анализатор именных групп [Сокирко, Толдова 2005].

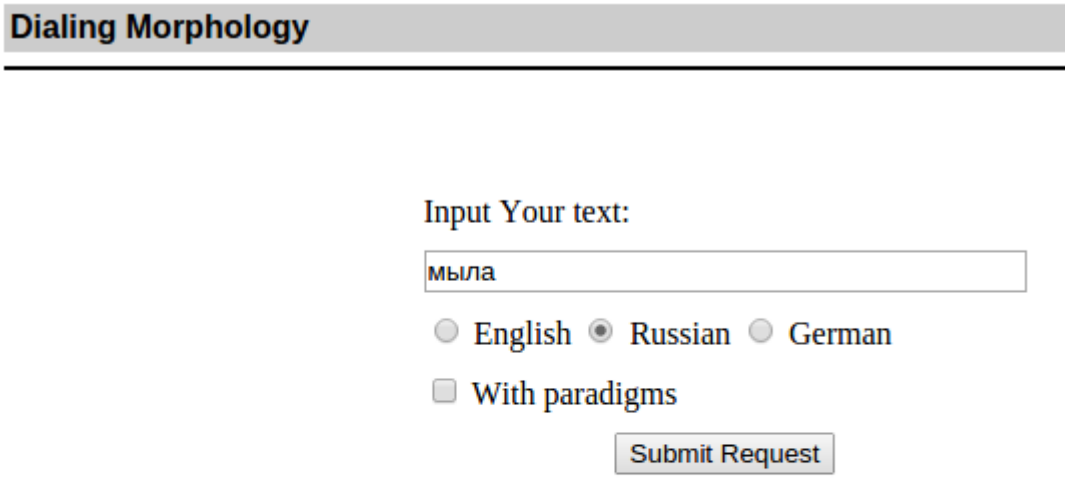
Модуль морфологического анализа АОТ реализован в виде библиотеки на языке C++ и сопровождается программой-редактором словарей (MorphWiz), позволяющей в удобном интерфейсе просматривать содержимое морфологического словаря, добавлять, удалять и исправлять описание слов.

Морфология в проекте АОТ включает в себя словари для русского (174 тысячи лемм), английского (104 тысячи лемм) и немецкого (218 тысяч лемм) языков. За основу русского морфологического словаря был взят грамматический словарь Зализняка. Все три словаря записаны в одинаковом формате, а поиск по ним осуществляется одним и тем же программным кодом. Демонстрационный интерфейс поиска по морфологическому словарю развернут на сайте проекта АОТ.

Морфологический модуль (включающий библиотеку LemmatizerLib) обрабатывает словоформы по отдельности и не учитывает их контекст. Результатом анализа словоформы является набор морфологических гипотез, каждая из которых включает следующие данные:

- флаг словарности, который показывает, основана ли гипотеза на словарной лемме или сгенерирована предиктивным алгоритмом.
- набор неизменяемых грамматических признаков (**граммем**), например одушевленность существительного или вид глагола;
- текстовая строка, представляющая лемму (аношкинские коды, о которых будет сказано чуть ниже);
- часть речи;
- множество наборов изменяемых грамматических признаков (граммем) для данной словоформы, по набору для каждого варианта лемматизации, например число существительного, род прилагательного и пр.

На рис. 1 показаны наборы граммем всех возможных форм для слова "МЫЛА" представлены в виде их объединения (рд, ед U им, мн U вн, мн = вн, рд, им, ед, мн).



Found	Dict ID	Lemma	Grammems	
+	но,	МЫЛО	С ср,вн,рд,им,ед,мн,	
+	пе,нс,	МЫТЬ	Г дст,прш,жр,ед,	

Перечень всех использующихся в проекте АОТ частей речи и граммем представлен на странице проекта. Уникальный двухбуквенный идентификатор, соответствующий некоторой комбинации значений селективных признаков и граммем, называется **аношкинским кодом** или **анкодом**. Например, коду *аа* соответствует комбинация граммем «С,мр,ед,им», а коду *аб* - «С,мр,ед,рд». Список всех возможных анкодов (сочетаний граммем), использующихся библиотекой LemmatizerLib, содержится в файлах {r,e,g}gramtab.tab (для русского, английского и немецкого языков).

Модель словоизменения в системе АОТ может порождать маловероятные (или ошибочные) словоформы. Например, возможно порождение форм сравнительной степени прилагательных и форм на "по-", образованных на основе форм сравнительной степени, в тех случаях, где эти формы не кажутся естественными: «призывнОй» - «призывнЕе»* - «попризывнЕе»*, «столОвый» - «столОвее»* - «постолОвее»* и т.д. Лишние формы не являются проблемой при анализе текста, однако могут вызвать сложности при постановке слова или словосочетания в указанную форму.

С технической точки зрения лингвистические анализаторы проекта АОТ спроектированы так, чтобы их было удобно использовать изнутри других программ: библиотека на C++ LemmatizerLib, COM-интерфейс Lemmatizer и .Net интерфейс LemmatizerNet. Для обработки одного текстового файла существует программа FileLem. Готовые к использованию модули и программы доступны на сайте проекта по адресу http://aot.ru/download.php.

На настоящий момент работа над лингвистическими модулями проекта АОТ продолжается, и морфологический словарь пополняется новыми словами.

4.3. Морфологический анализатор Pymorphy2 и словарь проекта OpenCorpora

Словарная база проекта АОТ была использована в других разработках, среди которых морфологический анализатор pymorphy на языке Python и морфологический словарь проекта OpenCorpora. С содержательной точки зрения pymorphy — это реализация алгоритмов морфологического анализа, описанных в документации проекта АОТ, на языке Python с

поправкой на внутреннее представление данных (в АОТ используется конечный автомат, в pymorphy — таблицы пар ключ-значение). Последняя версия анализатора была выпущена в 2011 году, и с тех пор разработка и исправление ошибок в pymorphy остановлены.

С 2012 года ведущий разработчик проекта pymorphy Михаил Коробов начал работу над новым проектом — pymorphy2, в котором заново реализованы хранение словаря и поиск по нему. В качестве словаря в этой системе используется словарная база проекта "Открытый корпус".

Так же как и морфологический анализатор АОТ, pymorphy2 обрабатывает словоформы независимо друг от друга. В качестве результата он возвращает набор гипотез, каждая из которых является возможной морфологической интерпретацией заданной словоформы. Рассмотрим пример из документации по pymorphy2:

```
>>> morph.parse('стали')
[Parse(word='стали',     tag=OpencorporaTag('VERB,perf,intr     plur,past,indc'),
normal_form='стать',     score=0.983766,     methods_stack=((<DictionaryAnalyzer>,
'стали', 884, 4),)),
Parse(word='стали',        tag=OpencorporaTag('NOUN,inan,femn        sing,gent'),
normal_form='сталь',     score=0.003246,     methods_stack=((<DictionaryAnalyzer>,
'стали', 12, 1),)),
Parse(word='стали',        tag=OpencorporaTag('NOUN,inan,femn        sing,datv'),
normal_form='сталь',     score=0.003246,     methods_stack=((<DictionaryAnalyzer>,
'стали', 12, 2),)),
Parse(word='стали',        tag=OpencorporaTag('NOUN,inan,femn        sing,loct'),
normal_form='сталь',     score=0.003246,     methods_stack=((<DictionaryAnalyzer>,
'стали', 12, 5),)),
Parse(word='стали',        tag=OpencorporaTag('NOUN,inan,femn        plur,nomn'),
normal_form='сталь',     score=0.003246,     methods_stack=((<DictionaryAnalyzer>,
'стали', 12, 6),)),
Parse(word='стали',        tag=OpencorporaTag('NOUN,inan,femn        plur,accs'),
normal_form='сталь',     score=0.003246,     methods_stack=((<DictionaryAnalyzer>,
'стали', 12, 9),))]
```

В этом примере pymorphy2 предложил шесть вариантов анализа словоформы *стали*. Первый вариант — это форма глагола *стать*. Остальные пять вариантов связаны с существительным *сталь* в единственном (sing) и множественном (plur) числе в родительном (gent), дательном (datv), предложном (loct), именительном (nomn) и винительном (accs) падежах

В pymorphy2 реализован предиктивный морфологический анализ для слов, которых не удалось найти в словаре. В следующем примере анализатор обработал слова *бутявковедами*, правильно определил его базовую форму *бутявковед* и понял, что это одушевленное существительное (NOUN,anim) мужского рода (masc), в форме множественного числае (plur) и творительного падежа (ablt):

```
>>> morph.parse('бутявковедами')
[Parse(word='бутявковедами',   tag=OpencorporaTag('NOUN,anim,masc   plur,ablt'),
normal_form='бутявковед',      score=1.0,      methods_stack=((<FakeDictionary>,
'бутявковедами', 51, 10), (<KnownSuffixAnalyzer>, 'едами')))]
```

В pymorphy2 также предусмотрена возможность постановки слов в начальную форму, произвольную форму по указанным граммемам и согласования существительного с заданным числительным.

Морфологический словарь проекта OpenCorpora, использующийся в pymorphy2, является побочным результатом работы над аннотированным Открытым корпусом текстов на русском языке OpenCorpora. В его разметке каждому слову сопоставляется не только набор граммем, описывающих его форму, но и номер леммы в морфологической базе данных. Морфологический словарь OpenCorpora основан на словаре АОТ и полностью его включает.

Набор граммем для этого словаря был разработан заново, после чего словарь АОТ был автоматически преобразован в новый формат. Словарь OpenCopora продолжает вручную пополняться словами, которые были встречены в текстах корпуса. На момент написания этой книги копия словаря со свежими дополнениями ежедневно публикуется на сайте проекта в текстовом формате и в формате XML. Проект OpenCorpora не имеет своего модуля морфологического анализа, pymorphy2 пока является единственным инструментом, который использует словарную базу Открытого корпуса.

4.4. Анализатор Mystem

Морфологический анализатор mystem — это внутренняя разработка компании Яндекс, и его исходный код недоступен. Бинарные сборки программных модулей mystem для Windows, Mac OS и Linux опубликованы на сайте Яндекса. Первая версия mystem была разработана Ильёй Сегаловичем и Виталием Титовым. На данный момент в свободном доступе находится третья версия, в состав которой включён модуль снятия частеречной омонимии. Неоднозначность между разными формами одного и того же слова не снимается.

Без снятия омонимии ./mystem -e UTF-8 -gni	Со снятием омонимии ./mystem -e UTF-8 -gn**d**
эти{этот}	эти{этот}
типы{тип}	типы{тип}
стали{становиться|сталь}	**стали{становиться}**
есть{быть|есть}	**есть{есть}**
в{в}	в{в}
цехе{цех}	цехе{цех}

Анализатор mystem использует словарь, который «зашит» внутрь программы и недоступен для пользователя: его нельзя ни посмотреть, ни изменить. Если возникает необходимость дополнить словарную базу анализатора специфической для некоторой предметной области лексикой, пользователи могут подключать свой дополнительный словарь к mystem при запуске. Подробное описание формата пользовательского словаря и способов использования mystem есть на сайте Яндекса.

При помощи mystem можно обработать текстовую строку или текстовый файл. Для использования анализатора изнутри других программ существуют несколько модулей, предоставляющих программный интерфейс к mystem. Среди них pymystem3 для Python и mystem-scala для Java и Scala. Эти модули разработаны и поддерживаются самостоятельными разработчиками, а их список ведётся на странице Mystem сайта NLPub.

Список доступных морфологических анализаторов для русского языка не так велик, как для английского, однако он не ограничивается перечисленными выше. Существуют также следующие морфоанализаторы для русского языка:

- TreeTagger — языконезависимый инструмент с закрытым исходным кодом, он действует на основе машинного обучения (деревья решений), хотя последние его версии помимо статистики начали использовать словарь, так что из чисто статистического анализатора он стал гибридным.
- Stemka — вероятностный морфологический анализатор для русского и украинского языков. Доступен в виде исходого кода на языке C++.
- морфологический модуль проекта FreeLing.
- набор правил стемминга для русского языка на языке обработки строк Snowball.

Компьютерные лингвисты, разрабатывающие морфологические парсеры для русского языка, принимают участие в соревновании по оценке качества морфоанализаторов и разработке стандартов морфологического анализа [Ляшевская и др. 2010]. В ходе

соревнования проводится тестирование систем, выполняющих обработку всех слов во входном тексте и работающих с контекстными данными. Соревнование проводится по нескольким дорожкам. В зависимости от возможностей решения задач автоматического разрешения морфологической неоднозначности, различаются два вида морфоанализаторов – сильные и слабые (проводящие или не проводящие разрешение неоднозначности). Для слабых парсеров введены дорожки «Лемматизация», «POS», «Морфология», «Редкие слова», а для сильных парсеров – дорожки «Дизамбигуация: леммы» и «Дизамбигуация: частеречные теги». Соревнование позволило создать «Золотой стандарт» морфологической разметки русскоязычных текстов: так, согласно экспертным оценкам, точность ручной разметки эталонного корпуса составляет 94,4% по леммам, 95,4% по частеречным тегам, 89,0% по тегам грамматических категорий и 85,5% по всей морфологической аннотации.

5. Статистические методы анализа слов

5.1. Статистическая частеречная разметка

Статистический подход к компьютерной морфологии основан на методах машинного обучения, о которых подробно рассказывается в пятой главе нашей книги. Кроме того, о них можно узнать из тех работ, которые приводятся в списке литературы в конце данной главы. А здесь мы подробно рассмотрим только один вид статистического анализа слов — **автоматическую частеречную разметку** (part of speech tagging, POS-tagging). В результате этой разметки каждому слову в предложении будет проставлена **метка** или **тег** (от английского слова *tag* — бирка) соответствующей части речи. Например, если на входе компьютер получит предложение *Дождь стучит в стекло*, то на выходе он должен выдать последовательность частей речи *существительное, глагол, предлог, существительное.* Или, если заменить существительное символом *N*, глагол символом *V*, а предлог символом *P*, то на выходе получится цепочка *NVPN*.

Как научить компьютер проводить автоматическую разметку? Казалось бы, надо дать ему словарь, где *дождь* и *стекло* будут отмечены как существительные, а *стучит* — как глагол. Но слово *стекло* может оказаться глаголом прошедшего времени, ведь язык неоднозначен. Кроме того, чисто статистические методы обходятся без словарей.

Тут можно вспомнить, что в русском языке после предлога будет идти, скорее всего, существительное, а не глагол. Поэтому при автоматической разметке надо учитывать не только тег каждого слова, но и тот контекст, в котором оказывается этот тег.

Проще всего, если у нас уже есть хороший размеченный корпус — большой набор текстов, где у каждого слова в предложении проставлен соответствующий ему тег:

Дождь стучит в стекло
N V P N

С помощью такого корпуса можно обучить компьютер размечать новые предложения.

Для начала на основе корпуса надо составить два множества:

1) Множество словоформ Ψ — набор всех слов этого корпуса в том виде, в каком они находятся в тексте. Ведь для компьютера слова *дождь* и *дождя* — это разные цепочки символов.

2) Множество тэгов Ω — набор всех тэгов, которые использовались в этом корпусе. Их может быть, например, два десятка, но вряд ли больше сотни, даже если привычные части речи разбить на несколько категорий.

Каждое предложение в нашем корпусе можно рассматривать как цепочку $x_1, x_2,\ldots x_n$. Такая цепочка состоит из *n* элементов множества Ψ, где *n* — длина предложения.

Соответствующие словам этого предложения теги тоже можно рассмотреть как цепочку, последовательность элементов из множества тегов Ω. Эту цепочку можно представить так: $y_1, y_2,\ldots y_n$.

Иными словами, можно сказать, что цепочке словоформ $x_1, x_2,\ldots x_n$ из множества Ψ соответствует цепочка тегов $y_1, y_2,\ldots y_n$ из множества Ω.

Задача статистического анализатора — найти для любой цепочки словоформ из множества Ψ **наиболее вероятную** последовательность тегов из множества Ω. Например, если у нас есть предложение *Пришел добрый пёс*, то для него теоретически возможны любые последовательности из трех тегов: *NNN, NVV, NPV* и все другие комбинации. Но компьютер должен прийти к выводу, что самая вероятная последовательность для данного предложения — это *VAN* (глагол, прилагательное, существительное).

Как научиться считать и сравнивать такие вероятности? В следующем разделе мы рассмотрим один способ решения этой проблемы.

5.2. Триграммная скрытая Марковская модель

Автоматическую частеречную разметку можно провести с помощью **триграммной скрытой Марковской модели** (Trigram hidden Markov model — Trigram HMM). Эта модель строится на основе корпуса. Для нее нам понадобятся упоминавшиеся в предыдущем разделе множества Ψ и Ω, а также два набора параметров, каждый из которых соответствует некоторой условной вероятности:

3) Параметры s(w|t)

4) Параметры q(t|u, v)

Параметр *s(w|t)* — это вероятность того, что тегу *t* соответствует слово *w*. Например, *s(пёс|N)* — это вероятность того, что тегу *N* (существительное) соответствует слово *пёс*. Казалось бы, логичнее использовать обратный параметр — вероятность появления тега *t*, если у нас есть слово *w*, но интересующий нас параметр именно такой — в качестве условия выступает тег, и оценивается вероятность соответствия ему того или иного слова.

С помощью имеющегося корпуса этот параметр легко оценить для всех словоформ w из множества Ψ и тегов t из множества Ω. Здесь нужно сделать важное замечание: с помощью корпуса мы не сможем *вычислить* вероятности чего-либо. Мы сможем только *оценить* вероятность, выбрать ее правдоподобное значение. Для параметра *s(w|t)* это можно сделать следующим образом:

1. Вначале нужно вычислить *c(t)* — сколько раз в корпусе встретился тег *t*. Например, сколько раз там встретился тег *N* (допустим, он встретился 10 000 раз).

2. Затем посчитать *c(t →w)* — сколько раз в корпусе тегу *t* соответствовала словоформа *w*. В нашем примере — сколько раз тегу *N* соответствовала словоформа *пёс* (допустим, 20 раз).

3. После этого можно посчитать *s(w|t)* как результат деления *c(t →w)* на *c(t)*. Это значение может рассматриваться как вероятность того, что тегу *t* будет соответствовать словоформа *w*:

$$s(w|t) = \frac{c(t \rightarrow w)}{c(t)}$$

В нашем примере $s(\textit{нёс}|N) = \frac{c(N \rightarrow \text{нёс})}{c(N)} = \frac{20}{10\,000} = \frac{1}{500}$

Оставшийся параметр *q(t|u, v)* — это вероятность появления тэга *t* при условии того, что перед ним находятся теги *uv*. Например, *q(N|V, A)* — это вероятность появления тега *N* при условии того, что перед ним находятся теги *VA*. Эта вероятность для русского языка довольно велика. Такая последовательность тегов встречается, например, в сочетании *съел невкусный суп* или в предложении *Произошла странная история.*

На основе корпуса этот параметр можно посчитать для всех возможных сочетаний тегов. Самый простой способ вычислить параметр *q(t|u, v)* похож на вычисление параметра *s(w|t)*.

1. Первым делом вычисляется *c(uvt)* — сколько раз в корпусе встречается цепочка из идущих подряд тегов *uvt*. В нашем примере — сколько раз встречалась цепочка тегов *VAN* (допустим, 200 раз).

2. Затем нужно посчитать *c(uv)* — сколько раз в корпусе встречается цепочка из идущих подряд тегов *uv*. В нашем примере — сколько раз встретились теги *VA* (допустим, 80 000 раз).

3. После этого *q(t|u, v)* вычисляется путем деления *c(uvt)* на *c(uv)*. Это соответствует вероятности того, что тег t будет идти после цепочки тегов *uv*. У нас получится 200/80000 = 1/400.

При вычислении параметра *q(t|u, v)* необходимо учитывать начала и концы предложений. Если какой-то тег встретился в самом начале предложения, то у него нет словоформ, которые могли бы выступить в роли *u* и *v*. Ничего страшного. В этом случае *u* и *v* можно заменить звездочками. Тогда параметр *q(N|*, *)* будет обозначать вероятность того, что тег *N* (существительное) встречается в абсолютном начале предложения, а параметр *q(N|*, A)* будет соответствовать вероятности того, что тег *N* (существительное) идет в предложении вторым после тега *A* (прилагательное).

Концы предложений тоже необходимо учитывать, поэтому последним тегом можно считать слово STOP. Оно обозначает, что за ним тегов уже нет — предложение закончилось. Тогда для нашей системы необходимо также почитать параметры *q(STOP|u, v)* — вероятности того, что последовательность тегов *u, v* — последняя в предложении.

Конечно, полученные значения вероятностей будут приблизительные. Их можно улучшить с помощью методов сглаживания, о которых будет рассказано в седьмой главе этой книги.

После того, как на основе корпуса посчитаны значения всех параметров, наша триграммная скрытая Марковская модель готова. С ее помощью можно оценивать вероятности соответствия предложения и цепочки тегов. Делать это можно по следующей формуле:

$$P(x_1, \ldots x_n, y_1, \ldots y_{n+1}) = \prod_{i=1}^{n+1} q(y_i|y_{i-2}, y_{i-1}) \prod_{i=1}^{n} s(x_i|y_i)$$

Слева — совместная вероятность последовательности словоформ и последовательности тегов. Справа — произведение двух множителей. Первый множитель — это произведение *n+1* параметра, вероятностей появления каждого тега y_i после двух предшествующих ему тегов. Тег y_{n+1} соответствует слову STOP. Второй множитель —

произведение параметров, отвечающих за соответствие тех или иных словоформ различным тегам.

Эта формула появилась не случайно, у нее есть объяснение и вывод, но мы не будем перегружать эту книгу обилием математических подробностей. Любопытный читатель может прочитать об этой формуле в специализированных учебниках. Кроме того, некоторые дополнительные пояснения приводятся в статистической части седьмой главы данной книги, посвященной машинному переводу.

Рассмотрим на примере, как работает эта формула. Какова вероятность того, что предложению *Пришел добрый пёс* соответствует цепочка тегов *VNN* (глагол, существительное, существительное). Да, мы сознательно рассмотрим пример с ошибкой: *VNN,* а не *VAN* (глагол, прилагательное, существительное).

Рассчитаем эту вероятность по формуле, не забыв поставить последним тегом слово *STOP*:

$$P(\text{Пришел, добрый, пёс, V, N, N, STOP}) = q(V\,|*, *) \times q(N\,|*, V) \times q(N\,|V, N) \times$$
$$\times\, q(\text{STOP}\,|N, N) \times s(\text{Пришел}|V) \times s(\text{добрый}|N) \times s(\text{пёс}|N)$$

Теперь нам остается посчитать эти вероятности для всех других последовательностей тегов (AVN, NVV, AAN и пр.), а затем выбрать наиболее вероятный вариант. Но если у нас 20 тегов, то для предложения из трех слов это будет 20^3=800 вариантов. А для предложения из 10 слов — 20^{10}=10240000000000 вариантов. Слишком много.

Тут на помощь приходит динамическое программирование, а точнее — **алгоритм Витерби** (Viterbi algorithm). Он позволяет решать такие задачи коротким путем, не перебирая все возможные варианты. В сети есть много его описаний, поэтому мы не будем здесь его подробно рассматривать, а перейдем к еще одной лингвистической проблеме.

5.3. Частеречная разметка незнакомых слов

Допустим, в новом предложении упоминается чиновник с фамилией *Неподкупнов*. Такой фамилии не было в обучающем корпусе, поэтому параметр *s(Неподкупнов|t)* не определен ни для какого тега. Как быть?

Одно из решений может быть таким. Очень редкие слова в обучающем корпусе заменяются псевдословами. Вот примеры возможных замен:

Варианты редких слов в обучающем корпусе	**Псевдослово для замены**
Сорокин, Черномор, Коваленко	Слово-с-большой-буквы
1963, 2018, 1147	Четыре-цифры
13, 55, 63	Две-цифры
ДЮШ, ВЦСПС	Все-прописные

После замены редких слов псевдословами проводится обучение системы — вычисляются описанные выше параметры. Теперь, если нам встретится в предложении чиновник Неподкупнов, который обучающем корпусе не попадался, его можно будет заменить на псевдослово (нашем случае — Слово-с-большой-буквы), и тогда он будет проанализирован как Сорокин, Черномор и Коваленко.

В заключение отметим, что самый надежный способ научиться компьютерной морфологии — это попытаться самостоятельно создать систему морфологической разметки. В этом вам помогут литература и ссылки на электронные ресурсы, которые приводятся в конце данной главы.

Литература

Зализняк А. А. Грамматический словарь русского языка. Словоизменение. М., 1977. Изд. 2-е, испр. и доп. М., 1980. Изд. 3-е. М., 1987. Изд. 4-е, испр. и доп. М., 2003. Изд. 5-е, испр. М., 2008. URL: http://zaliznyak-dict.narod.ru/index.htm

Коваль, С.А. Лингвистические проблемы компьютерной морфологии. СПб., 2005.

Леонтьева Н.Н. Автоматическое понимание текстов. Системы, модели, ресурсы. М., 2006.

Ляшевская О.Н. и др., Оценка методов автоматического анализа текста: морфологические парсеры русского языка. Компьютерная лингвистика и интеллектуальные технологии: По материалам ежегодной Международной конференции «Диалог–2010». Вып. 9(16). М., 2010.

Сокирко А.В. Морфологические модули на сайте www.aot.ru // Компьютерная лингвистика и интеллектуальные технологии: Труды международной конференции «Диалог'2004». М., 2004. URL: http://www.dialog-21.ru/Archive/2004/Sokirko.pdf.

Сокирко А. В. Сравнение эффективности двух методик снятия лексической и морфологической неоднозначности для русского языка (скрытая модель Маркова и синтаксический анализатор именных групп) / А. В. Сокирко, С. Ю. Толдова // Интернет-математика 2005. Автоматическая обработка веб-данных. М., 2005. С. 80-94. URL: http://www.aot.ru/docs/RusCorporaHMM.htm

Шаров С. А., Беликов В. И., Копылов Н. Ю., Сорокин А.А., Шаврина Т. О. Корпус с автоматически снятой морфологической неоднозначностью: К методике лингвистических исследований. Компьютерная лингвистика и интеллектуальные технологии. По материалам ежегодной конференции Диалог. М., 2015. URL: http://www.dialog-21.ru/digests/dialog2015/materials/pdf/SharoffSAetal.pdf

Brill E. A simple rule-based part-of-speech tagger // Proceedings of the Third Conference on Applied Natural Language Processing (ANLF-92), Trento, Italy, 1992.

Brian Roark, Richard William Sproat. Computational approaches to morphology and syntax. Oxford University Press, 2007.

Clark Alexander. et al. (Eds.) The Handbook of Computational Linguistics and Natural Language Processing. Wiley-Blackwell, Malden, MA, 2010. URL: http://stp.lingfil.uu.se/~santinim/sais/ClarkEtAl2010_HandbookNLP.pdf

Hajič J. Disambiguation of Rich Inflection (Computational Morphology of Czech). Prague, 2004.

Hajič J., Krbec P., Květoň P., Oliva K., Petkevič V., Serial Combination of Rules and Statistics: A Case Study in Czech Tagging. ACL, 2001.

Jurafsky Daniel, James H. Martin. Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Second Edition. Upper Saddle River, NJ, 2009. URL: https://www.cs.colorado.edu/~martin/slp2.html

Karlsson F., et al. (Eds.) Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text. Mouton De Gruyter, 1995.

Manning Christopher D., Schütze Hinrich. Foundations of Statistical Natural Language Processing. MIT Press. Cambridge, MA, 1999.

Mitkov Ruslan (Ed.). The Oxford Handbook on Computational Linguistics. Oxford University Press, 2005.

Protopopova E. V., Bocharov V. V. Unsupervised learning of part-of-speech disambiguation rules // Компьютерная лингвистика и интеллектуальные технологии: По материалам ежегодной Международной конференции «Диалог» (Бекасово, 29 мая — 2 июня 2013 г.). Вып. 12 (19). — М., 2013.

Schmid H. Part-of-speech tagging with neural networks //COLING '94 - Proceedings of the 15th conference on Computational linguistics. Volume 1. Stroudsburg (PA): ACL, 1994. Pp. 172-176. URL: http://portal.acm.org/citation.cfm?id=991915

Электронные ресурсы

AOT http://aot.ru/, репозиторий проекта http://sourceforge.net/projects/seman/
FileLem http://sourceforge.net/p/seman/svn/HEAD/tree/trunk/Source/FileLem/
FreeLing, морф. модуль http://nlp.lsi.upc.edu/freeling/doc/userman/html/node5.html
MULTEXT-East http://nl.ijs.si/ME/, http://corpus.leeds.ac.uk/mocky/.
MyStem, перечень известных программных интерфейсов https://nlpub.ru/Mystem
Mystem-Scala для Java и Scala https://github.com/alexeyev/mystem-scala
OpenCorpora http://opencorpora.org
Pymorphy (репозиторий проекта) https://github.com/kmike/pymorphy
Pymorphy2 https://github.com/kmike/pymorphy2
Pymystem3 для Python (https://github.com/Digsolab/pymystem3)
Snowball http://snowball.tartarus.org/
Stemka, исходный код на языке C++ http://www.keva.ru/stemka/stemka.html
TreeTagger https://nlpub.ru/TreeTagger
TRmorph https://github.com/coltekin/Trmorph
Викисловарь в скачиваемом виде http://dumps.wikimedia.org/ruwiktionary/latest