Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра системного программирования

Семантическое автодополнение

Курсовая работа студента 445 группы

Удалова Александра

Научный руководитель:

Д. В. Хитров

Санкт-Петербург

2011

Оглавление

Введение

Мотивацией к созданию данной работы послужило желание ускорить набор текста на мобильных устройствах с сенсорным экраном.

Ввод текста на этих устройствах заметно медленнее, чем ввод с настоящей клавиатуры ПК, особенно если учесть время на допущенные ошибки при вводе и их исправление. Встроенный в некоторые мобильные ОС механизм автокоррекции ошибок часто допускает ошибки и этим ещё более раздражает пользователя.

В связи с этим возникла идея при наборе текста отображать пользователю в реальном времени подсказки в виде нескольких возможных окончаний набранной пользователем части слова (этот процесс будем называть автодополнением). Предполагая, что пользователь обычно набирает осмысленный текст, слова можно подсказывать, учитывая семантику уже набранного текста. Появилось несколько идей, какие именно особенности языка и как можно учитывать при подсказке следующего слова.

Обзор

Целью данной курсовой работы является исследование существующих методов автодополнения слов естественных языков, а также разработка собственной системы автодополнения на основе изученных идей и алгоритмов.

Работа была выполнена вместе с моим одногруппником, поэтому далее в этом разделе последует краткое описание всей проделанной работы, а в следующих разделах – отчёт о конкретно моих результатах.

Сперва было решено сделать прототип работающей системы. Разработка логики системы велась на Java. Для ручного тестирования был создан минимальный пользовательский интерфейс в виде одной веб-страницы. В качестве языка для автодополнения был выбран английский.

Основная идея работы прототипа состояла в следующем: в тексте на английском языке существуют некоторые закономерности расположения слов относительно друг друга, по которым можно предсказать следующее слово на основе нескольких последних. Это утверждение можно упростить до следующего: в тексте на английском языке различные n-граммы (n-грамма – набор из n подряд идущих слов) встречаются с различной частотой, и на основе информации об этих частотах и последних n – 1 словах можно сделать вывод о вероятности появления в качестве n-ого слова некоторого слова из словаря. Например, после слов `in a timely` по статистике чаще всего идёт слово `manner`. Как выяснилось далее, эта мысль была обширно исследована в [1], доказала свою состоятельность и в итоге стала основополагающей и в нашей курсовой работе.

Далее свод знаний о языке в виде списка утверждений «эта n-грамма встречается в текстах с такой частотой» будем называть корпусом (лингвистический корпус – это несколько другое, но схожее понятие). Для работающего прототипа понадобилась некоторая база знаний о языке, и решено было составить корпус. Для этого было написано приложение, которое открывает текстовый файл, читает находящийся в нём текст, разбивает его на предложения и, проходя по каждому предложению, обновляет частоты встречаемости соответствующих n-грамм. Приложение было запущено на нескольких книгах из [2], полученные сведения были объединены в один файл. Следует отметить, что в качестве формата для хранения корпуса был выбран текстовый формат файла, при котором в каждой строчке файла указано n слов через пробел и одно целое число, обозначающее сколько раз эта n-грамма встретилась в рассматриваемых текстах. Текстовый формат (вместо возможного использования реляционных баз данных) был выбран в связи с простотой восприятия

человеком (нужно было как-то проверить адекватность полученных результатов), а также в связи с простотой и скоростью его обработки самим приложением.

После создания первой версии корпуса 1-грамм (который представлял из себя просто список, сколько раз каждое слово встретилось в разобранных текстах) был разработан прототип, который работал следующим образом: пользователь вводит в командной строке предложение, приложение разбивает его на слова, берёт последнее введённое слово в качестве префикса для подсказки, затем перебирает все слова из корпуса в порядке убывания их частоты встречаемости, и выводит 10 (или менее) первых слов, которые начинаются на приведённый префикс.

Следующей идеей было использовать для подсказок простейшую вариацию метода «part of speech tagging», который в общем случае позволяет понять, к какой части речи относится каждое слово, введённое пользователем. Для этого из свободных источников ([3]) была раздобыта база (в виде текстового файла), в которой для каждого более-менее используемого слова английского языка было описано, какими частями речи оно бывает представлено в тексте. Для улучшения алгоритма подсказок было использовано следующее соображение: после некоторых слов чаще всего идут слова конкретной части речи, и поэтому стоит считать слова этой части речи более вероятными в качестве кандидатов. Например, после слова `this` чаще всего идёт существительное или прилагательное, поэтому некоторые из них вероятно имеет смысл позиционировать выше в итоговой подсказке, чем некоторые, даже часто по недоразумению встречающиеся, глаголы. Для составления подобной базы (опять же, в текстовом виде) к приложению, составляющему корпус, был приписан механизм, который считает, какие части речи с какой частотой встречаются после каждого разобранного слова. Как оказалось, это не дало ощутимого улучшения. Стало ясно, что чтобы более-менее улучшить алгоритм, понадобится помимо анализа частей речи производить синтаксический анализ введённой части предложения, чтобы выяснить, какой член предложения может идти следующим. Этот процесс слишком сложен (как описано в ч. 3 [1]), поэтому от подобного метода было решено отказаться. Исходные коды, тем не менее, сохранены и они находятся в package `old` в приложенных исходниках (`Old.java`, `OldVocabulary.java`).

Было решено воспользоваться методом статистического вывода (statistical inference, гл. 6, ч. 2 [1]). В общем случае этот метод, обладая некоторыми знаниями о случайном процессе, на основе нескольких завершённых экспериментов оценивает вероятность исхода следующего эксперимента. В применении к автодополнению, в качестве случайного

процесса выступает набор пользователем n-грамм слов, а эксперимент – это каждое следующее набранное слово. На основе корпусов из n-грамм для различных n этот метод позволяет оценить вероятность появления в качестве следующего слова любого слова из словаря. Основные классы, относящиеся к алгоритму, расположены в package `algos`. Поскольку метод статистического вывода оставляет свободным выбор оценивающей функции, были испробованы различные её варианты (описанные в гл. 6 ч. 2 [1]): Laplace's law, Lidstone's law, Good-Turing estimation (впоследствии было решено остановиться на последней).

Для получения более обширного и качественного корпуса вместо того экспериментального, что был создан в начале, был использован свободный проект [4]. Было выкачано большое количество информации, преобразовано и представлено в виде корпуса для приложения. В итоге для системы были готовы корпуса из 1-грамм, 2-грамм, 3-грамм. Набор скриптов и исходных файлов, ответственных за разбор этих данных, находится в папке `gcorpus`.

Так как корпус был составлен на основе художественной и научной литературы, стало понятно, что ограничиться применением только его для автодополнения обычной бытовой речи недостаточно. Из предположения о том, что каждый человек по-своему изъясняется (а именно, есть набор слов, который человек использует часто, и логично использовать это знание при подсказке), был реализован механизм создания корпуса каждого пользователя системы, который был совмещён с уже реализованными частями из `algos`. Исходные файлы, относящиеся к многопользовательской системе – класс `algos.UserEngine`, а также package `users`.

Чтобы улучшить разобранный корпус засчёт бытовой речи, было решено использовать то место всемирной сети, в котором люди общаются больше всего, а именно социальные сети. В частности, социальная сеть Twitter предоставляет удобный API для разработчиков клиентских приложений. На примере Twitter была попытка использовать предложения, которые люди пишут в trending topics, для построения нового, «современного» и более бытового корпуса. Эта попытка оказалась неудачной, потому что, как оказалось, в Twitter далеко не все посты из trending содержат актуальную и полезную информацию. Исходные коды представлены в package `twitter`.

Была разработана тестирующая система для того, чтобы определить, что выбрать в качестве оценивающей функции, улучшить константы, и в целом убедиться в преимуществе выбранного статистического метода перед первоначальным, синтаксическим, решением. Её

исходный код целиком представлен в `Test.java`.

В дополнение к интерфейсу, доступному из командной строки, было реализовано простейшее веб-приложение на основе AJAX.

Обработка корпуса

В июле 2009 года Google Labs опубликовало проект Books Ngram Viewer ([4]), в котором представило данные, полученные из проекта Google Books. Эти данные представляют из себя набор текстовых файлов, разбитых на части, в каждой из которых содержится набор n-грамм (для n от 1 до 5) с указанием, сколько каждая из них встречалась в книгах, доступных на Google Books, и в каком году. Было решено использовать эту информацию для создания собственного корпуса.

Из-за огромного объёма информации мы решили построить только корпуса 1-грамм, 2-грамм и 3-грамм. Возникла необходимость в реализации приложения, которое бы скачало разбитые на части базы n-грамм, отобрало из них некоторую разумную часть и сложило бы в новую базу результаты. Оно было успешно создано и применено, его исходные коды доступны в папке `gcorpus`.

Краткое описание файлов из папки `gcorpus`:

- `collect` – основной скрипт. Написан на языке bash и предназначен для запуска на ОС семейства UNIX. В качестве параметров ему необходимо передать n (какие n-граммы выкачивать), с какой и по какую части. Скрипт скачивает все указанные части, каждую часть разархивирует, затем запускает приложение `pystsch`, которое отбирает из n-грамм только существенные, и складывает результаты в отдельную папку.
- `pystsch.c` – исходный код приложения на языке C, отбирающего n-граммы из одной части. В качестве параметров ему необходимо передать имя файла. Все параметры являются константами и задаются до компиляции:
  - `HASH_PRIME` – простое число, используемое для хеширования (вместо сравнения n-грамм из соседних строк сравниваются их хеши, что быстрее);
  - `MIN_OCCURENCE` – минимальное суммарное число встреч n-граммы, необходимое для её отбора в результат;
  - `MIN_YEAR` – год, с которого учитываются n-граммы (необходимо, чтобы отсечь устаревшие слова и языковые выражения)
- `merge.cpp` – исходный код приложения на языке C++, соединяющего обработанные результаты каждой части в одну базу. Параметры те же, что и у `collect`. Его необходимо запустить один раз после того, как скрипт скачает и обработает все части. Оно выводит результат в текстовый файл, в котором n-граммы отсортированы по убыванию частоты и для каждой указано, сколько раз она встретилась.

Корпус на основе Twitter

Идея использовать социальные сети возникла из двух соображений:

- в социальных сетях люди используют разговорный язык, который может оказаться полезнее для системы, чем художественный;
- существуют события, происходящие в данный момент, о которых люди пишут часто (таким образом, система получит временной контекст)

В социальной сети Twitter ([5]) существует раздел trending topics («тренды»), содержащий темы, наиболее обсуждаемые в данный момент. Twitter предоставляет удобный API для разработчиков клиентских приложений ([6]), в частности, есть возможность получить список трендов, после чего по данному тренду можно сделать поиск и получить несколько последних сообщений.

Было создано приложение на Java, которое делало именно это. Его исходные коды представлены в package `twitter`. Назначение каждого класса более-менее очевидно из его названия и интерфейса. Также использовался стандартный package `org.json`, который требуется подключать самостоятельно.

Результаты, полученные приложением, оказались плачевны. В частности, среди наиболее часто встречающихся 3-грамм в трендах, помимо типичных для английского языка, не удалось найти ничего, относящегося к современным происходящим событиям. Я объясняю это тем, что, вероятно, во-первых, о каждом тренде пользователь рассказывает по-своему, используя собственные слова и выражения, а во-вторых, пользователи очень часто пишут вещи, не имеющие прямого отношения к тренду, или же как минимум эту связь тяжело отследить на уровне 3-грамм. Надежда на появление новых частых 3-грамм из разговорного языка также не оправдалась, вероятно потому что разговорный, а именно язык общения в интернете, гораздо богаче на различные формы и написания слов (включая ошибочные). Поэтому от создания корпуса на основе трендов решено было отказаться.

Тестирование

Было реализовано тестирующее приложение для того, чтобы убедиться в применимости реализованного метода семантического автодополнения. Его исходный код представлен в файле `Test.java`.

По историческим причинам все параметры, необходимые для работы приложения, такие как пути входных и выходных файлов, используемые оценочные функции, коэффициенты, проставляются прямо в исходном коде перед компиляцией. Для работы приложения, помимо корпусов, необходим файл с текстом. В данном случае система тестировалась на некоторой произвольной свободной книге середины XX века (с [2]). Текст разбивается на предложения, а в каждом предложении рассматривается пара из n-граммы и некоторого префикса следующего за ней слова. Системе автодополнения на вход даются эти n-грамма и префикс, и проводится подсчёт нескольких величин: сколько всего проведено попыток; сколько раз на каждой позиции (от 1 до 10) подсказки оказалось правильное слово; какова минимальная длина префикса слова, по которому система угадает его на первом месте, и другие.

Было проведено несколько экспериментов над системой, которая использует корпуса из 1-грамм, 2-грамм и 3-грамм с оценочной функцией Гуда-Тьюринга и некоторыми разумными коэффициентами (далее в таблицах ”1g,2g,3g+users”), в противовес «несемантической» системе (”1g”) – которая использует только корпус из 1-грамм (иными словами, это первоначальный подход, который сортирует слова по убыванию частоты встречаемости в языке и возвращает несколько первых, начинающихся на данный префикс). Также была рассмотрена система, запущенная без пользовательского режима (”1g,2g,3g”).

Результаты экспериментов представлены в двух таблицах ниже. В столбцах указаны два часла через запятую – первое значит длину префикса, предоставленного системе, второе минимальная длина слова, для которого этот тест запускался. Например, последний тест ”3,8” означает, что системе предлагалось дополнить все возможные n-граммы из предложений исходного текста, такие, что в следующем слове как минимум 8 букв, и при этом ей давался префикс из 3 букв.

В первой таблице показаны частоты попадания правильного слова на первое место в подсказке.

алгоритм	3,4	2,3	1,2	3,8
1g	41.88%	34.37%	25.41%	9.41%
1g,2g,3g	55.43%	48.04%	38.49%	24.10%
1g,2g,3g+users	55.58%	48.33%	38.70%	25.02%

Во второй таблице показаны частоты попадания правильного слова на одно из первых трёх мест в подсказке (тест возник из первоначальной цели курсовой и он обусловлен опытом набора текста на мобильном устройстве, при котором было бы вполне удобно видеть в качестве подсказки правильное слово на первых трёх местах).

алгоритм	3,4	2,3	1,2	3,8
1g	61.70%	52.79%	42.36%	22.63%
1g,2g,3g	73.20%	64.61%	55.88%	42.76%
1g,2g,3g+users	73.58%	64.92%	56.29%	44.08%

Как и ожидалось, система автодополнения на основе корпусов из 1-грамм, 2-грамм, 3-грамм оправдала свои ожидания и показала себя гораздо лучше несемантической системы.

В следующей таблице показано, каково среднее значение минимального количества букв, необходимых для отгадывания слова длины от 2 до 10 (показа его на первой позиции в подсказке).

алгоритм	2	3	4	5	6	7	8	9	10
1g	1.58%	1.55%	2.92%	3.67%	4.48%	5.07%	5.89%	6.40%	7.11%
1g,2g,3g	0.99%	1.20%	2.14%	3.33%	3.75%	3.93%	4.33%	5.69%	5.70%

По этой статистике семантическая система также заметно лучше. Тот факт, что она работает не медленнее несемантической, оправдывает её применение в реальности.

Реализация веб-интерфейса

Для удобства ручного тестирования был разработан и выложен в общий доступ веб-интерфейс к системе. Его основой является сервер, написанный на Java и использующий разработанную систему автодополнения для ответа на поступающие ему запросы. Его исходный код представлен в `Server.java`. Был использован простейший веб-сервер из пакета `com.sun.net.httpserver`.

Пользовательская часть представлена в папке `html`. Здесь находятся файлы:

- `php.php` – простейшая страничка на PHP, которая позволяет ввести текст в поле, запрашивает у сервера подсказку и выводит ответ
- `index.html` – простейшая страничка на HTML, состоящая из одного элемента `textarea`, которая позволяет получать подсказки сразу во время написания текста
- `js.js` – скриптовая часть этой страницы, реализованная на JavaScript с применением общедоступной библиотеки jQuery ([7]) и технологии JSONP для кросс-доменного запроса JSON (это позволило расположить всю клиентскую часть на отдельном домене, имя которого человек способен запомнить, в отличие от IP компьютера, на котором запущен сервер)

Используемые источники

1. Christopher D. Manning and Hinrich Schütze. 1999. Foundations of Statistical Natural Language Processing. Cambridge, MA: MIT Press.
   http://nlp.stanford.edu/fsnlp/promo/
2. Project Gutenberg: free eBooks
   http://www.gutenberg.org/wiki/Main_Page
3. Kevin's Word List Page
   http://wordlist.sourceforge.net/
4. Google Labs: Books Ngram Viewer
   http://ngrams.googlelabs.com/
5. Twitter
   http://twitter.com/
6. Twitter API Documentation
   http://dev.twitter.com/doc
7. jQuery
   http://jquery.com/