Московский Государственный Университет имени М.В.Ломоносова
Факультет Вычислительной Математики и Кибернетики
Кафедра Системного Программирования

Курсовая работа

Автоматическая оценка последовательности изложения научных текстов.

Автор:
группа 327
Манджиев Айта Викторович

Научный руководитель:
Недумов Ярослав Ростиславович

Москва, 2017

Содержание

Введение

В современном мире существует огромное количество научных статей практически на любую тему. Помимо этого новые научные работы появляются все чаще и чаще, поэтому ученому, который хочет изучить новую для себя область, порой приходится прочитать довольно много некачественно написанных и тяжело воспринимаемых статей.

Сложность работы со статьями не ограничивается только чтением, также очень сложно написать качественную научную работу. Тем не менее, научные статьи — это очень важное средство общения для людей, занимающихся наукой.

Именно поэтому разрабатываются множество разнообразных средств, которые помогают людям в написании качественные работ и в поиске хороших научных статей.

Один из методов, который можно использовать для оценки качества статьи — это показатель текучести (fluidity) статьи, представленный в [1]. Текучесть в общем относится к тому, насколько легко данный открывок текста может быть прочитан и понят. Текучий текст имеет ряд премуществ таких, как быстрое чтение, лучшее восприятие информации, бóльшая вероятность быть опубликованным. Также текучесть уменьшает количество интерпретаций, которые получает читатель от текста. А в случае научных статей это довольно важный фактор — донести до читателя именно то, что имел в виду сам автор и ничего другого.

Данный показатель используется в программном средстве, которое называется SWAN (Scientific Writing Assistant). А алгоритм, вычисляющий текучесть статьи описан в [1]. В нашей курсовой работе предлагается реализация вычисления текучести статьи и исследование вопроса о том, есть ли взаимосвязь между успехом статьи на конференциях и её текучестью.

1 Постановка задачи

Целью данной курсовой работы является исследование существования зависимости между текучестью статьи и качеством статьи с точки зрения её успеха на конференциях (получала ли она какие–то награды и на конференциях какого уровня она участвовала).

Необходимо реализовать метод оценки текучести (fluidity), предложенного в [1], где он представлен в виде псевдокода.

Необходимо собрать тестовый набор данных, состоящий из статей, которые разбиты на два класса: «хорошие» и «плохие». «Хорошими» и «плохими» они являются с точки зрения уровня конференций, на которых данные статьи были представлены, и были ли они среди победителей номинации «Лучшая работа».

После этого необходимо оценить текучесть тестовых статей и на основе этих данных обучить классификатор. Посредством кросс-валидации оценить точность данного классификатора и сделать вывод о существовании или отсутствии связи текучести и качества статьи.

2 Обзор существующих решений

2.1 SWAN —Scientific Writing Assistant

SWAN — компьютерная программа, основанная на правилах, которые включают в себя метрики оценки качества текста и обработку естественного языка. Данная программа анализирует те части научной статьи, которые производят первое впечатление на читателя — заглавие, аннотация, введение и заключение.

SWAN не дает общую оценку для статьи, а лишь выделяет проблемные места, которые могут быть улучшены с точки зрения метрик, использующихся в ней. Метрики, которые используются в данной программе, описаны в работе [1].

2.1.1 Метрики, использованные в SWAN

Вместо того, чтобы оценивать грамматику языка или правописание, SWAN анализирует ключевые элементы статьи. Вдобавок данный инструмент оценивает текучесть написания и структуры статьи.

В SWAN используется два вида метрик: автоматические и ручные. Примером автоматической оценки может быть поиск в статье выражений–паттернов и предложений, которые находятся в пассивном залоге. В качестве ручных используются те метрики, которые являются сложными для автоматической оценки. Такие как выделение ключевых слов заглавия или поиск предложения аннотации, которое является основным. Вот некоторые метрики:

- Проверка того, что аннотация и заглавие согласуются.
- Все ключевые слова заглавия должны встречаться в структуре статьи (из заголовков и подзаголовков) так, что она становится очевидна.
- Оценка текучести текста.

2.1.2 Текучесть в SWAN

Текучесть — одна из основных метрик, которые используются в SWAN. Но полностью автоматизировать оценку текучести текста довольно сложно. Из-за того, что оценка текучести основана на понимание текста читателем. Поэтому ручная оценка текучести

превосходит автоматическую, которая основана на понятиях темы и акцента, разработанных в [2]

Для вычисления упомянутых понятиий темы и акцента в SWAN используется Stanford parser. Тема — информация, которая связывает текущее предложений с предыдущим, а акцент — новая, важная информация, затронутая данным предложением. Акцент предложения, как правило, становится темой следующего предложения. На основе этих понятий SWAN оценивает тему и акцент последовательно идущих предложений, а также где именно в предложении они находятся. В зависимости от этого предложение может иметь один из следующих типов текучести:

- Текучее (Fluid) — предложение, связанное с предыдущим.
- Инвертированная тема (Inverted topic) — предложение, связанное с предыдущим, но эта связь находится только в самом конце предложения.
- Не синхронизированное (Out-of-sync) — предложение, связанное с предшествующим предложением, но между ними есть бессвязные предложения.
- Бессвязное (Disconnected) — предложение, которое не связано с предшествующими.

SWAN использует текучесть для выдачи предупреждений пользователю в случае, если он использовал «переходные выражения» (выражения, которые устанавливают искуственную связь, например, in addition, on the other hand и т.д.) в предложениях, которые не являются текучими. Также она выделяет предложения, которые могут нарушать текучесть и дает советы как эту проблему можно исправить.

К сожалению, исходного кода программы нет в открытом доступе.

2.2 Выводы

В обзоре был описан инструмент по работе с научными статьями SWAN. А также то, каким образом данный инструмент используется в расчете текучести текста.

Из получившегося обзора можно сделать вывод, что мы не можем воспользоваться данным инструментом для решения поставленной задачи. Так как, во-первых, он не предоставляет общую информацию по количеству предложений в каждом из классов текучести, а, во-вторых, исходный код программы получить мы не можем.

3 Исследование и построение решения задачи

Задача реализации метрики для оценки текучести (fluidity) текста и исследования зависимости между качеством статьи и её текучестью. разбивается на следующие подзадачи:

- Предварительная обработка текста.
    1. Разбиение текста на параграфы и предложения.
    2. Пометка каждого слова соответствующей ему частью речи.
    3. Удаление незначимых коротких выражений и ссылок на литературу.
- Разметка исходного текста.
    1. Построение дерева зависимостей для каждого предложения. Нахождение главных частей предложения и их существительных.
    2. Формирование набора слов темы (topic) и акцента (stress) для каждого предложения.
    3. Определение типа связи предложения на основании полученных наборов слов.
- Обработка полученной разметки.
    1. Подсчет общего количества предложений.
    2. Подсчет количества предложений в каждом классе текучести.
    3. Нормировка полученных данных.
- Формирование тестовой выборки.
    1. Формирование набора «хороших» и «плохих» статей.
    2. Применение алгоритма поиска текучести к данным статьям.
- Анализ полученного классификатора.
    1. Применение кросс-валидации для набора статей, полученного на предыдущем этапе.
    2. Обучение и анализ полученной точности классификатора.

3.1 Предварительная обработка текста

Перед тем, как применять алгоритм определения текучести, нам необходимо привести текст к формату, удобному для анализа. А именно — получить список предложений и слов в этих предложениях.

Первое, что нужно сделать — определить каким образом абзацы отделяютя друг от друга. Так как данная подзадача не является основной целью работы, будем считать, что абзацы отделаются друг от друга символом переноса строки. Благодаря такому соглашению мы можем получить список параграфов с помощью регулярных выражений.

В случае разбиения предложений ситуация осложняется тем, что мы не можем принять такое же соглашение как и для абзацев. Поэтому для разбиения текста на предложения и формирования списка этих предложений мы воспользуемся средствами библиотеки nltk tokenize, так как они используют уже обученную модель, которая показывает достаточно хорошие результаты.

На следующем шаге нам необходимо для каждого слова определить его часть речи. Воспользуемся библиотекой Stanford POS Tagger, которая является одной из лучших для решения данной задачи. Так как у нас уже есть список предложений, нам остается лишь применить методы библиотеки к ним.

Для того, чтобы не обрабатывать данные, которые создают шум при оценке текучести текста, удалим ссылки на литературу и незначимые короткие выражения (short stubs). При удалении ссылок выведем их общий вид и воспользуемся регулярными выражениями, а поиск незначимых коротких выражений производится путем обычного поэлементного сравнения.

3.2 Разметка исходного текста

На основе обработанного текста, полученного из предыдущего этапа, нам необходимо произвести окончательную разметку предложений по категориям текучести, представленным в [1].

В связи с тем, что алгоритм работает с главными и подчиненными частями предложений, нам необходимо отделить их друг от друга. В этом нам может помочь дерево зависимостей. Для его построения воспользуемся одним из лучших парсеров — Stanford parser. После применения данного парсера мы получим деревья зависимостей для пред-

ложений.

По полученному на предыдущем шаге дереву зависимостей мы можем найти подчиненные части предложения. Для этого мы ищем поддеревья с определенными тегами, которые помечают подчиненные части (SBAR, SBARQ). После того как мы нашли все такие поддеревья, мы их удаляем — в остатке получаем главную часть предложения.

Для алгоритма поиска текучести, который мы используем также необходимо найти подлежащие главной части предложения. Для этого воспользуемся уже упоминавшимся Stanford parser. На основе данного парсера можно найти подлежащие двумя способами — на основе универсальных зависимостей (Universal Dependencies) или на основе дерева зависимостей. Так как у нас уже есть дерево зависимостей вполне логично использовать его для поиска. Для реализации выбранного способа необходимо произвести поиск поддеревьев с тегом S таких, что среди прямых потомков корня этого поддерева есть «существительная фраза« (noun phrase, тег NP) и «глагольная фраза» (verb phrase, тег VP), тогда упомянутая «существительная фраза» и будет являться подлежащим.

Формирование наборов слов темы и акцента происходит на основе сопоставления существительных текущего предложения с наборами слов из предыдущего предложения. Сначала рассматриваются существительные из главной части, в результате сопоставления которых мы находим «сильный» набор слов темы. После этого поиск продолжается и рассматривает существительные всего предложения для того, чтобы найти «слабый» набор слов темы и набор слов акцента также «сильного» и «слабого». Так как первое предложение абзаца не может быть проанализировано на основе предыдущих (абзацы рассматриваются независимо), то для него мы задаем стандартный набор слов (тема — существительные из главной части предложения, акцент — остальные существительные).

В самом простом случаем мы можем определить тип текучести, если в предложении встречаются так называемые текучие слова (fluid words), которые должны встречаться до первого спряженного глагола (conjugated verb). В этом случае мы помечаем предложение как текучее (fluid) и задаем ему стандартные наборы слов темы и акцента. Иначе — после того, как мы нашли тему и акцент для предложения, мы можем определить его тип текучести на основе того, как эти наборы слов были найдены для текущего предложения. Конкретный алгоритм в виде псевдокода можно посмотреть в [1].

3.3 Обработка полученной разметки

На данном шаге предполагается подготовка данных в удобном формате для обучения классификатора.

Во–первых, подсчитаем количество предложений для каждого из классов текучести обрабатываемой статьи. Во–вторых, нормализуем их путем деления количества предложений каждого из классов на общее количество предложений. Нормализация производится для того, чтобы классификатор учитывал только соотношения между классами текучести и не брал во внимание непосредственное количество предложений в статье, так как оно может достаточно сильно разниться.

Таким образом мы получим данные о каждой статье, которые нормализированы от 0 до 1.

3.4 Формирование тестовой выборки

Для того, чтобы обучить и проанализировать классификатор нам необходимо составить выборку статей, на которых он будет обучаться. Так как предполагается бинарная классификация особо много статей для тестовой и тренировочной выборок не потребуется.

Поиск подходящих «хороших» и «плохих»будем производить на сайтах международных конференций (такие как SIGMOD, KDD, SIGIR). Рейтинг конференций можно посмотреть, например, здесь. Набор «хороших» статей будем формировать из статей, которые получили награду «Лучшая работа» (Best paper) на одной из конференций, которая имеет ранг A*. В то время, как набор «плохих» статей будет формироваться из обычных статей, которые были представлены на конференциях ранга C.

3.5 Обучение и анализ полученного классификатора

С целью обучения классификатора мы сформировали выборку из ста статей с соотношением 1:1 «хороших» и «плохих» статей. Так как такого количества статей не достаточно для полноценного разделения на обучающую и тестовую выборки воспользуемся кросс-валидацией для оценки качества работы классификатора.

Воспользуемся библиотекой sklearn как одной из самых популярных и качественных библиотек по машинному обучению. Поскольку выборка статей достаточно невелика

воспользуемся классификатором «GradientBoostingClassifier», так как он достаточно хорошо работает на небольших объемах входных данных.

Для начала применим кросс-валидацию для данных, которые не были нормированы. В данном случае получаем результаты, которые отражены в таблице 1.

В первом столбце описывается название классификатора, который использовался. Второй столбец таблицы указывает на сколько блоков делились исходные данные при применении кросс-валидации. В третьем столбце находится информация о том, была ли исходная выборка нормализована или нет. В четвертый стобец записан средний результат примененной кросс-валидации, который считается как среднее арифметическое, взятое от результатов, полученных на каждом из блоков.

Классификатор	Количество частей разбиения	Нормализация данных	Средняя точность
GradientBoostingClassifier	3	нет	0.65
GradientBoostingClassifier	3	да	0.49
GradientBoostingClassifier	4	нет	0.67
GradientBoostingClassifier	4	да	0.5
GradientBoostingClassifier	5	нет	0.67
GradientBoostingClassifier	5	да	0.52

Таблица 1: Результаты применения кросс-валидации к классификатору

В таблице 2 представлены данные по среднему значению соотношений между классами текучести для хороших и плохих групп статей. А также дисперсии по указанным классам.

Тип группы статей	Fluid	Inverted topic	Out-of-sync	Disconnected
Хорошие	0.2	0.16	∼0.01	0.64
Плохие	0.22	0.18	∼0.01	0.6
Дисперсия хороших	0.067	0.056	0	0.1
Дисперсия плохих	0.062	0.058	0	0.097

Таблица 2: Средние соотношения между количеством статей в каждом из классов и их дисперсии

Из полученных результатов видно, что классификатор работает довольно плохо на нормализованных данных. Так как средняя точность, которую показывает классификатор — 0.5. А также соотношение между текучими и не текучими предложениями в плохих статьях в среднем даже лучше, чем в хороших. Стоит заметить, что дисперсия у хороших и плохих статей примерно одинкова.

3.6 Выводы

Из всего вышесказанного можно сделать вывод, что прямой взаимосвязи соотношения между количеством предложений в каждом из классов текучести статьи и качеством статьи нет. Относительно хороший результат на ненормированных данных получился, вероятнее всего, из-за того, что «хорошие» статьи в выборке состояли примерно из одного количества предложений, в то время, как общее количество предложений в «плохих» статьях варьировалось.

4 Описание Экспериментальной части

4.1 Обоснование выбранного инструментария.

В качестве языка реализации был выбран Python, так как для данного языка программирования разработано много библиотек по работе с естественным языком, а это как раз именно то, что нам и нужно. Также задача не требует высокой производительности, а значит непосредственная работа с памятью не потребуется.

Для разбиения текста на предложения, разметки слов частями речи и построения деревьев разбора были использованы средства библиотеки по обработке естественного языка nltk. А именно:

- tokenize — пакет для разбиения текста.
- Stanford POS Tagger — программный продукт, который считывает текст на одном из поддерживаемых языков и обозначает каждое слово его частью речи.
- Stanford parser — парсер естественного языка, который работает с грамматической структурой предложения.
- sklearn — библиотека, предоставляющая средства по работе с машинным обучением.

4.2 Общая схема работы.

Диаграмма, которая показывает общую схему работы представлена на рисунке 1.

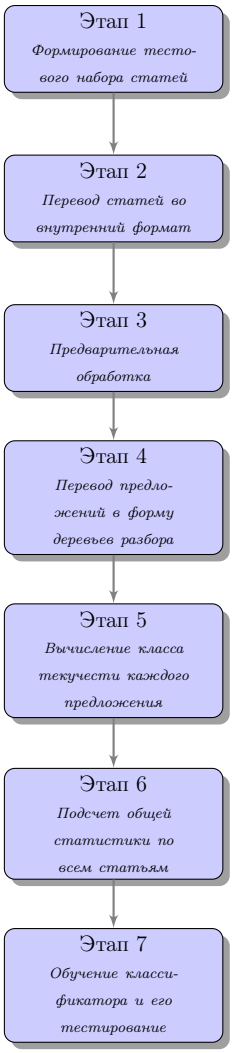

Рис. 1: Общая схема работы

Перед непосредственным анализом статей статей необходимо для начала сформировать тестовую выборку. Как уже упоминалось в третьей главе, мы будем извлекать статьи согласно рейтингу конференций, на которых они были представлены, так как мы хотим выяснить существует ли взаимосвязь между качеством статьи и её текучестью.

После того, как тестовая выборка сформирована нам необходимо перевести все статьи в формат удобный для анализа. В нашей программе мы будем использовать следующий формат — все параграфы слиты в одну строку (то есть символ перевода строки «\n» встречается только между параграфами), предложения разделяются точкой.

Далее мы подаем на вход программе название файла, в котором находится статья в текстовом виде. Все статьи анализируются отдельно друг от друга. Все результаты

записываются в единый файл для будущего анализа.

На первом шаге происходит считывание статьи и разбиение её на параграфы и предложения. Для удобства будем хранить параграфы в качестве списка, в то время как каждый параграф сам является списком, состоящим из предложений.

Список параграфов, полученный на предыдущем этапе, далее передается функции, которая производит предварительную обработку данных. А именно — удаление всех ссылок на литературу, разметка слов соответствующими частями речи и удаление коротких незначимых выражений (short stubs).

Для удаления ссылок мы воспользовались библиотекой regex путем вывода общего вида ссылок на литературу и удаления всех подстрок, удовлетворяющих данному шаблону.

Далее необходимо пометить все слова соответствующей им частью речи. В этом нам поможет Stanford POS Tagger. Заметим, что при использовании библиотеки nltk, желательно размечать все предложения разом, так как иначе данный «теггер» будет разворачивать виртуальную джава–машину и производить разметку на ней при каждом вызове метода разметки, что может занять значительное количество времени.

После получения размеченного текста легко найти и удалить короткие незначимые выражения, так как исходные предложения уже разбиты на отдельные слова.

На данном этапе нам необходимо получить дерево разбора. Его мы строим с помощью Stanford parser. После этого посредством анализа каждому предложению мы приписываем класс текучести, к которому он принадлежит. Все это делается на основе псевдокода, описанного в [1], с помощью языка программирования Python.

В итоге результаты работы алгоритма (а именно — количество предложений в каждом из классов текусти) записываются в файл.

Для запуска программы для каждой статьи из тестовой выборки был написан скрипт на языке bash.

На основе полученных данных мы можем обучить классификатор и проверить его точность с помощью метода кросс–валидации. В этих целях была использована библиотека sklearn.

4.3 Общая архитектура системы.

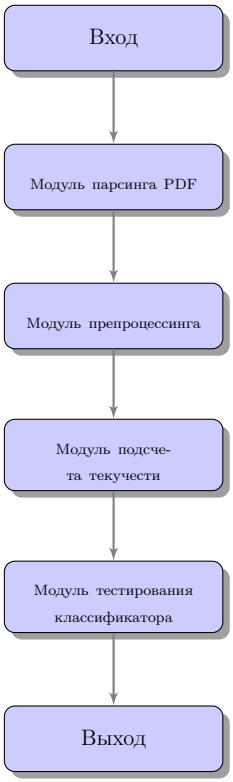

Рис. 2: Общая архитектура системы

Общая архитектура системы изображена на рисунке 2.

Модуль парсинга PDF отвечает за перевод статьи из формата файла pdf в текстовый формат. При этом удаляются ненужные части статьи, например — список используемой литературы или титульный лист.

Модуль препроцессинга обрабатывает текстовый формат статьи и приводит его к удобному для анализа представлению.

Модуль подсчета текучести производит непосредственный расчет количества предложений в каждом из классов текучести.

Модуль тестирования классификатора отвечает за обучение классификатора на основе данных, полученных путем применения алгоритма ко всем статьям выборки, и применение метода кросс-валидации для определения точности полученного классификатора.

4.4 Характеристика функционирования.

Метод кросс-валидации был применен на выборке, состоящей из 100 статей. 50 из которых были выбраны в качестве «хороших» и 50 в качестве «плохих».

Количество символов	Общее время работы алгоритма	Среднее время работы препроцессинга	Время разметки частей речи	Время построения дерева разбора
31032	61	12.4	11.9	47.6

Таблица 3: Средние значения работы алгоритма на статьях. (секунды)

В таблице 3 приведены средние значения отработки алгоритма на статьях из тестовой выборки. Все значения указаны в секундах.

Заключение

В рамках данной курсовой работы был реализован алгоритм вычисления текучести текста, который был описан в форме псевдокода в [1]. В данном алгоритме использовались Stanford POS Tagger и Stanford parser для построения и работы с деревьями разбора предложений.

Был собран набор данных, состоящий из статей, получивших награду «Лучшая работа» на ведущих конференциях, а также из обычных статей, принимавших участие в каких-либо конференциях. Все эти статьи были приведены к формату, удобному для анализа вышеупомянутым алгоритмом.

Был построен классификтор, на основе которого применялась кросс-валидация для исследования существования зависимости между текучестью статьи и её успехом на конференциях. Данный классификатор был протестирован на нормализованных входных данных и ненормализованных.

В ходе работы было показано, что прямая зависимости между успехом статьи на конференциях и соотношением между количеством предложений в каждом из классов текучести отсутствует.

Список литературы

[1] *Turunen T.* Introduction to Scientific Writing Assistant (SWAN) —Tool for Evaluating the Quality of Scientific Manuscripts. — 2013.

[2] *Gopen. G. D.* Expectations: Teaching Writing From The Reader's perspective. — 2014.

[3] *Kinnunen T.* SWAN-scientific writing AssistaNt: a tool for helping scholars to write reader-friendly manuscripts. — 2012.

Приложение