Содержание

Введение

В последнее десятилетие социальные сети и Интернет-СМИ стали важным источником обратной связи для бизнеса и, отчасти, электронной коммерции. Популярность контента новостных сайтов распределена крайне неравномерно, некоторые новости получают непропорционально большую долю внимания со стороны пользователей. Прогнозирование популярности новостей является актуальной задачей для владельцев новостных сайтов, рекламных агентств и других медиа компаний, поскольку это напрямую влияет на ценообразование для доступа к информации, размещения рекламы и объявлений.

Популярность новостей может быть подсчитана разными способами: она может определяться количеством просмотров новости пользователями, количеством комментариев которые собрала новость, а также числом ссылок на данную новость. Популярность новостей формируется в результате взаимодействия многих факторов, прежде всего, таких как: качество содержания, востребованность, способ продвижения, взаимное влияние пользователей. В данной работе изучались методы использующие факторы, которые могут быть вычислены для прогнозирования новостей *априори*, то есть, до момента их публикации. Такими факторами (признаками) являются текстовое содержание новости, автор, планируемая дата публикации и информация о ранее опубликованных на данном сайте новостях. Для прогнозирования популярности использовались алгоритмы машинного обучения (в частности, методы регрессии), где численное значение популярности выступало в роли зависимой переменной, а перечисленные признаки представленные определенным образом в роли объясняющей. Одной из основных задач при этом было представление текстов новостей в виде вещественных векторов с которыми работают алгоритмы регрессии.

При анализе текстовой информации наиболее часто используется векторная модель, в которой тексты представляются в виде вещественных векторов фиксированной длинны. Значение каждой компоненты в таких векторах характеризует наличие или отсутствие определенного слова в тексте. Однако, существует ряд ситуаций когда не удается эффективно применять упомянутую *классическую* векторную модель, поскольку она не достаточно точно описывает закономерности содержащиеся в исходных данных. Такие ситуации возникают если в текстах ис-

пользуются имена собственные, устойчивые выражения, а также из-за омонимии. Применяя различные методы обработки естественных языков, можно получить улучшение точности модели, что, как правило, приводит к увеличению качества регрессии. Предложенный подход, с использованием алгоритмов обработки естественного (таких как, определение частей речи и выявление *именованных сущностей*), был использован в реализованной системе с целью улучшения точности прогнозирования.

Постановка задачи

Цель данной дипломной работы — исследовать различные методы регрессии и протестировать их эффективность в задаче прогнозирования популярности новостей. При этом возникают следующие подзадачи:

1. Сбор данных. Необходимо реализовать систему для автоматизации сбора данных из новостных источников и социальных сервисов. Система должна обеспечивать возможность сбора актуального на сегодняшний день объема данных – сотни тысяч и миллионы документов (веб-страниц с новостями) за срок порядка нескольких дней.

2. Преобразование текстов в векторную модель. Система должна автоматически выполнять преобразование собранных данных в *векторную модель*.

3. Исследовать возможные расширения и улучшения данной модели с помощью алгоритмов обработки естественных языков.

4. Построить регрессионную модель для прогнозирования, и оценить качество результатов.

IDEF0 диаграмма, моделирующая процессы, которые требуется автоматизировать для решения первых трех подзадач находится в *приложении «А»*.

1. Обзор предметной области

Процедура автоматического прогнозирования (prediction) популярности новостей включает две основные части: представление текстов новостей в виде векторов признаков и построение регрессионной модели на созданном массиве векторов. В обзоре подробно рассматривается векторная модель представления текстовых документов из области информационного поиска [1]. Необходимость преобразования коллекции текстовых документов в векторную модель определяется тем обстоятельством, что все методы регрессии, использованные в данной работе, требуют, чтобы входные данные были представлены в виде последовательностей чисел одинакового размера и одинакового формата. Далее в обзоре рассматривается задача восстановления регрессии (regression estimation [2]) и различные методы оценки качества итогового алгоритма. В конце приведен краткий обзор использованных источников.

1.1. Векторная модель пространства

Пусть задано множество документов (текстов) D и множество терминов (слов) T. В классической модели документ рассматривается как вектор, размерность которого равна количеству терминов в *словаре*, а каждый компонент вычисляется с помощью некоторой функции ставящей в соответствие паре «термин-документ» значение “важности” (веса) термина в контексте данного документа. Рассматриваемый в данной работе метод определения “важности” основывается на использовании двух основных идей.

Частота встречаемости термина. TF (от англ. — term frequency) — отношение числа вхождения некоторого термина к общему количеству терминов документа. Таким образом, оценивается важность термина t_i в пределах отдельного документа:

$$TF = \frac{n_i}{\sum_k n_k}, \tag{1.1}$$

где n_i есть число вхождений термина в документ, а в знаменателе — общее число терминов в данном документе. В этом случае порядок слов перестает иметь значение. Например, такие тексты, как: “Паша быстрее, чем Саша” и “Саша быстрее, чем Паша” становятся идентичны в рамках этой модели.

Инвертированная частота документа. Использование только частоты термина в документе недостаточно, так как не все термины документа одинаково важны в контексте документа: например, слово “трубопровод” дает больше информации о тематике документа, чем, например, союз “или”; в коллекции документов в области автомобильной индустрии, слово “авто” будет встречаться практически в каждом документе. Другими словами, разные термины по-разному важны для документа, даже если они встречаются в документе одинаково часто. Можно было бы использовать частоту термина в коллекции (CF) для определения понижающего коэффициента для тех терминов, которые слишком распространены в коллекции и имеют малое влияние на релевантность. Однако, общепринятым решением является использование частоты документа (DFT). Чтобы использовать частоту документа, вводится специальная величина — инвертированная частота документа

IDF (от англ. — inverse document frequency), которая определяется следующим образом:

$$IDF = \log \frac{|D|}{|d_i \supset t_i|}, \tag{1.2}$$

где $|D|$ — количество документов в корпусе (коллекции текстов), $|d_i \supset t_i|$ — количество документов, в которых встречается t_i (когда $n_i \neq 0$). Отметим, что IDF редко встречающегося термина высокий, а IDF часто встречающегося термина низкий. В рассматриваемой весовой схеме, называемой $TFIDF$, веса терминам назначаются в соответствии с формулой:

$$TFIDF_{t,d} = TF_{t,d} \times IDF_t \tag{1.3}$$

и отвечают следующим характеристикам:

1. Наибольший вес имеют термины, встречающиеся много раз, но в небольшом количестве документов.

2. Меньший вес имеют термины, которые либо меньше встречаются в документе, либо встречаются в большем количестве документов.

3. Наименьший вес имеют термины, которые встречаются практически во всех документах.

Теперь мы можем рассмотреть каждый документ как вектор, размерность которого равна количеству терминов в словаре, а каждый компонент вычисляется по формуле (1). Вектора документов, вычисленные по приведенной схеме будем обозначать $\vec{V}(d)$. Коллекция документов (корпус) может быть рассмотрена как множество векторов в едином векторном пространстве, в котором каждая ось отвечает за один термин. Как уже упоминалось ранее, что такое представление “теряет” порядок следования терминов в текстах документов.

В рамках данной модели можно определить способ вычисления меры близости двух векторов, принадлежащих векторному пространству. Простой идеей является модуль разности двух векторов, но этот метод не подходит по следующей причине:

два документа с очень похожим содержанием могут сильно различаться в данной мере только потому что один из них намного длиннее другого (относительные частоты встречаемости терминов одинаковы, но их абсолютные величины сильно различаются). Чтобы компенсировать эффект длины документа, стандартным решением является вычисление косинуса угла между векторами $\vec{V}(d_1)$ и $\vec{V}(d_2)$:

$$sim(d_1, d_2) = \frac{\vec{V}(d_1) \times \vec{V}(d_2)}{|\vec{V}(d_1)| \times |\vec{V}(d_2)|} \tag{1.4}$$

где числитель представляет собой скалярное произведение векторов, а знаменатель - произведение модулей векторов. Теперь, имея документ d, можно найти все похожие документы — такая функциональность требуется, например, для метода регрессии по алгоритму K-ближайших.

1.2. Задача восстановления регрессии

Задано множество объектов X (в нашем случае – вектора документов), множество допустимых ответов Y (значения популярности документов), и существует *целевая функция* (target function) $y^* : X \to Y$ значения которой $y_i = y^*(x_i)$ известны только на конечном подмножестве объектов $\{x_i, \ldots, x_l\} \subset X$. Пары «объект-ответ» (x_i, y_i) называются *прецедентами*. Совокупность пар $X^l = (x_i, y_i)_{i=1}^l$ называются *обучающей выборкой* (training sample).

Задача *обучения по прецедентам* [3] заключается в том, чтобы по выборке X^l восстановить зависиммость $y*$, то есть построить *решающую функцию* (decision function) $a : X \to Y$, которая приближала бы целевую функцию y^* причём не только на объектах обучающей выборки, но и на всём множестве X. Если $Y = \mathbb{R}$, то это задача называется задачей восстановления регрессии (regression estimation).

Моделью алгоритмов называется параметрическое семейство отображений

$$A = \{g(x, \theta) | \theta \in \Theta\}, \tag{1.5}$$

где $g : X \times \Theta \to Y$ — некоторая фиксированная функция, Θ — множество допустимых значений параметра θ, называемое пространством параметров или пространством поиска (search space).

Функционал качества алгоритма a на выборке X^l:

$$Q(a, X^l) = \frac{1}{l} \sum_{i=1}^{l} L(a, x_i) \tag{1.6}$$

$L(a, x)$ — это неотрицательная функция, характеризующая величину ошибки алгоритма a на объекте x, называемая функцией потерь (loss function). Если $L(a, x) = 0$, то ответ $a(x)$ называется корректным. В задачах регрессии наиболее часто используются следующие функции потерь:

- $L(a, x) = |a(x) - y^*(x)|$ — отклонение от правильного ответа; функционал Q называется средней ошибкой алгоритма а на выборке X^l;

- $L(a,x) = (a(x) - y^*(x))^2$ — квадратичная функция потерь; функционал Q называется средней квадратичной ошибкой алгоритма a на выборке X^l.

Классический метод обучения, называемый минимизацией эмпирического риска (empirical risk minimization, ERM), заключается в том, чтобы найти в заданной модели A алгоритм a, доставляющий минимальное значение функционалу качества Q на заданной обучающей выборке X^l:

$$\mu(X^l) = \arg\min_{a \in A} Q(a, X^l) \tag{1.7}$$

Стоит заметить, что в задаче восстановления регрессии с n числовыми признаками и квадратичной функцией потерь метод минимизации эмпирического риска есть ничто иное, как метод наименьших квадратов:

$$\mu(X^l) = \arg\min_{\theta} \sum_{i=1}^{l} (g(x_i, \theta) - y_i)^2 \tag{1.8}$$

Пока ещё не создан универсальный метод обучения по прецедентам, способный решать любые практические задачи одинаково хорошо [3]. Каждый метод имеет свои преимущества, недостатки и границы применимости. На практике приходится проводить численные эксперименты, чтобы понять, какой метод из имеющегося арсенала лучше подходит для конкретной задачи. Обычно для этого методы сравниваются по скользящему контролю (cross validation). В данной работе тестирование проводилось с помощью экспериментов на реальных данных для этого фиксировалась задача, и к ней применялись несколько методов регрессии при различных значениях параметров.

1.3. Обзор использованных источников

В [2, 3] приводится подробное теоретическое описание алгоритмов машинного обучения использованных в данной работе, методов их настройки и тестирования. В книге [1] дается введение в информационный поиск и приводится исчерпывающее описание векторной модели пространства и методов ее оптимизации (уменьшения размерности). В статьях [5, 6, 7, 8] исследуется возможность прогнозирования индексов (таких как S&P500) и котировок фондовых рынков через анализ текстовых сообщений в социальной сети *Twitter* и записей в финансовых блогах. В статье [4] исследуется эффективность метода прогнозирования популярности контента сайта *Digg.com*, используя стохастические модели социальной динамики. Статья [9] описывает систему *Ediscope* и результаты ее применения, в данном исследовании изучалась зависимость между словами в заголовках новостей и колличеством ссылок на эти новости в социальных сетях *Twitter* и *Facebook*. Книги [10, 11] посвящены решению практических задач обработки естественных языков с помощью языка программирования *Python* и библиотеки *NLTK*, которая использовалась в данной работе на этапе построения словаря термов и преобразования текстов в векторную модель. Статья [12] посвящена методу отбора значимых признаков в задачах классификации текстов с помощью алгоритма *LASSO* (Least Absolute Shrinkage and Selection Operator).

2. Выбор средств разработки

2.1. Язык программирования

В качестве основного языка программирования был выбран язык *Python*. Существует ряд причин, по которым был сделан такой выбор:

- Язык имеент мощный и лаконичный синтаксис. Язык поддерживает несколько парадигм программирования: процедурную, объектно-ориентированную и функциональную. Синтаксис языка поддерживает такие возможности, как: лямбда выражения, списочные выражения (list comprehensions), замыкания (closures), "срезы" для обращения к коллекциям, декораторы и д.р. Как правило, язык позволяет писать код, в несколько раз более короткий чем языки с C-подобным синтаксисом что дает потенциально меньше мест для совершения ошибки.

- Кросс-платформенность: код не содержащий специфичных C-расширений может быть запущен на любой поддерживаемой платформе: Linux, BSD, OS X, Windows NT и др.

- Автоматическое управления памятью позволяет исключить класс проблем связанный с утечкой памяти. При этом существует вероятность что сборщик мусора не сможет вовремя освободить неиспользуемые программой ресурсы (память, дескрипторы файлов и т.д.) и программа завершиться аварийно. Обычно такие ситуации возникают при большой нагрузке и решаются форсированием сборки мусора внутри кода программы.

- Развитая стандартная библиотека содержит большое колличество структур данных и алгоритмов необходимых для эффективной реализации прикладных задач. Имеется классический набор структур данных — контейнеров (массивы, списки, деревья, хеш-таблицы и т.д.), библиотеки для объектной работы с потоками и процессами, библиотека для работы с HTTP, модульного тестирования и другие. Особенно стоит отметить широкий набор средств для работы со строками.

- В дополнение к стандартной библиотеке существует огромное количество сторонних библиотек — Python занимает третье место по количеству про-

ектов хранящихся в системе *Github*, ставшей стандартом *де-факто* для хранения проектов с открытым исходным кодом. Также язык активно поддерживается такой компанией как *Google*. Краткий обзор использованных библиотек представлен далее.

- Отсутствие необходимости компиляции и наличие REPL существенно сокращает время на разработку и тестирование. При этом ценой за интерпретацию кода является сравнительно невысокая производительность программы. Тем не менее стоит учитывать что активные вычисления приходятся на небольшой процент кода и могут быть ускоренны с помощью расширений на языке ANSI C. Также существуют специальные интерпретаторы содержащие JIT компиляторы и имеющие большую скорость выполнения: PyPy и Cython.

- Наличие пакетного менеджера управляющего программными пакетами и их зависимостями существенно облегчает развертывание системы на новых компьютерах.

В целом, язык Python был выбран за удобный синтаксис, позволяющий писать и изучать сторонний, как правило, легкочитаемый код. Большой выбор доступных библиотек, позволяющий создавать эффективные решения на всех уровнях реализации системы: от работы с потоками и математикой до веб приложений и решения задач машинного обучения.

2.2. Обзор используемых библиотек

В ходе реализации данной работы использовались сторонние библиотеки:

1. Для реализации распределенной обработки данных был выбран фреймворк Celery. Celery проще, имеет меньше зависимостей и требований чем другой доступный фреймворк Disco, который более предназначен для обработки очень больших объемов данных (применяется, например, корпорацией Nokia). Была реализована парадигма Map-reduce и Celery использовался для обеспечения обмена сообщениями в распределенной среде. Map-Reduce [13] была выбрана как наиболее часто используемая в задачах распределенной обработки данных и абстрагирующая программиста от задач ручного контроля за ходом выполнения параллельных процессов. Map-reduce (Apache Hadoop) используется, например, в упомянутой системе Ediscope.

2. Для работы с HTML была использована библиотека *lxml*, которая обеспечивает низкоуровневый, объектный интерфейс к DOM-дереву страниц, а также реализует выполнение запросов на языке XPath. Lxlm имеет больше возможностей и является более производительной чем библиотека beautifulsoup.

3. Для задач обработки естественного языка использовалась библиотека NLTK которая содержит в себе основные алгоритмы: сегментация, определение частей речи, стемминг, лемматизация, поиск именованных сущностей, тематическая классификация текстов и др. NLTK являются наиболее развитой и часто используеммой и содержит подробную документацию с большим количеством примеров, книги [10, 11] и активное сообщество разработчиков.

4. Для решения задач регрессии использовалась библиотека. *scikit-learn*. Scikit-learn позволяет решать следующие задачи: обучение с учителем (supervised learning), выбор статистических моделей (model selection), обучение без учителя (unsupervised learning) и имеет наибольший арсенал алгоритмов среди прочих подобных библиотек языка Python — PyML, PyBrain, Shogun и д.р.

5. Для реализации графического интерфейса использовался веб-фреймвовк *django*, ORM-подсистема которого послужила также основой для классов-моделей разработанного решения.

UML диаграмма разработанных классов представлена в *приложении «Б»*.

3. Сбор данных

Как уже упоминалось выше, тестирование алгоритмов проводилось на реальных данных, собранных с сайтов *Forbes.com* и *BussinessInsider.com*, поэтому первой задачей стоял сбор коллекции текстов новостей необходимых для обучения моделей. Эта задача включает в себя три подзадачи: поиск ссылок на страницы с новостями, загрузка HTML документов по найденным ссылкам, извлечение текста новостей и других интересующих нас признаков из загруженных HTML документов. Далее рассмотрим все три подзадачи подробнее.

3.1. Поиск ссылок внутри новостных источников

Большинство новостных сайтов имеет двухуровневую структуру: *лента новостей* (*news river*) в которой, как правило, содержатся только заголовки с ссылками на новости и их краткие описания и *страницы новостей* – содержащие полные тексты новостей. Система сбора данных, разработанная в рамках данной работы предполагает что пользователь должен ввести шаблон адресов страниц с новостными лентами и запрос на языке *XPath* который позволял бы извлечь ссылки из HTML страницы с лентой. Построение запроса осуществляется вручную, однако не представляет большой сложности, поскольку существует инструменты позволяющие автоматизировать этот процесс. Рис. 3.1 иллюстрирует построение такого запроса для сайта *businessinsider.com* с помощью браузера *Mozilla Firefox* и расширения *XPath Checker*.

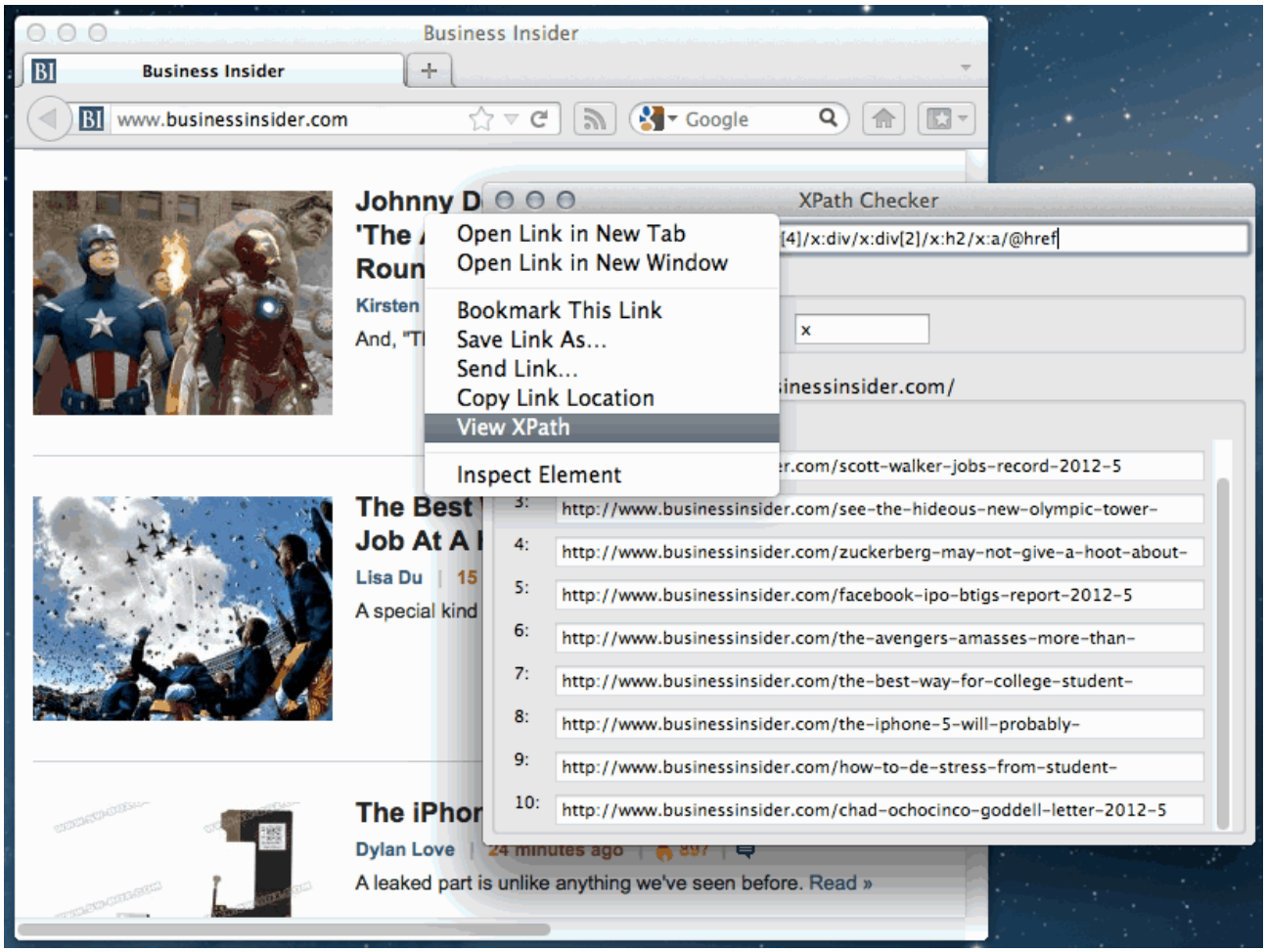


Рис. 3.1: Постоение запроса *XPath*

После того, как построен *XPath* и получен шаблон адресов страниц с лентами новостей, система использует их чтобы собрать указанное число лент и извлечь из них ссылки на отдельные страницы с новостями. На этом этапе используется библиотека *lxml* которая помимо остальных возможностей, также реализует исполнение запросов *XPath*. Для проверки *XPath* была реализованная процедура тестирования, которая брала заданную выборку загруженных страниц и пытается применить к ним запрос *XPath*, после этого выдает пользователю количество страниц на которых запрос отработал некорректно — например, не извлек необходимого количества ссылок (этот критерий можно использовать на практике, поскольку все ленты одного сайта как правило имеют одинаковое количество новостей).

3.2. Загрузка HTML страниц и извлечение текста

Для выполнения процедуры загрузки HTML было реализовано решение использующее парадигму Map-Reduce [13] для выполнения параллельной обработки дынных. В качестве фреймворка для Map-Reduce использовалась библиотека программная Celery. Рассмотрим основные стадии в рамках данной парадигмы.

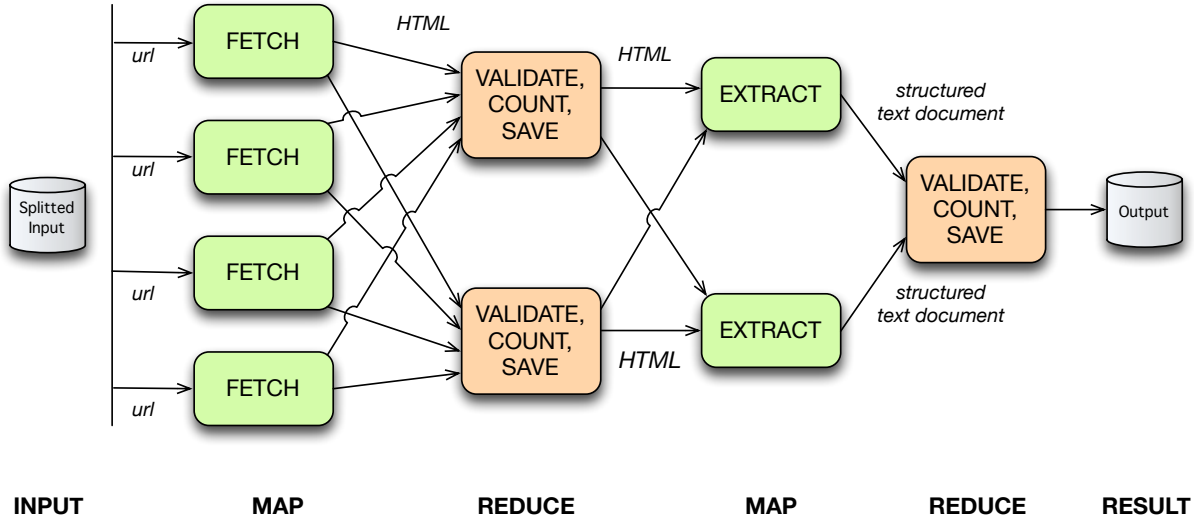


Рис. 3.2: Схема *Map-Reduce* процесса сбора данных

Процесс сбора состоит из следующих стадий выполняющихся параллельно:

1. **INPUT** Ввод входных данных. На этом шаге главный процесс (мастер, *master*) получает входные данные (ссылки, *URL*) группирует их по несколько единиц и передает другим процессам.

2. **MAP** Процессы-рабочие (воркеры, *workers*) получают наборы ссылок и загружают их из Интернет, отправляют полученные данные обратно главному процессу.

3. **REDUCE** Главный процесс получает загруженные HTML страницы, делает проверку (сайты, с которых запрашиваются данные могут вместо нужных страниц возвращать коды ошибок). Если данные прошли проверку, они сохраняются в БД и передаются для дальнейшей обработки. В противном случае, мастер-процесс инициализирует повторную загрузку по ссылкам, которые не прошли проверку, при этом ведется ведется учет таких повторных загрузок и при преодолении определенного порога ссылка “выбрасывается” (такое происходит если ссылка — *мертвая*).

4. **MAP** Из страниц, прошедших проверку извлекаются признаки: заголовок новости, автор, дата публикации, полный текст новости и число просмотров. Подробнее этот процесс будет описан в следующем подразделе.

5. **REDUCE** Аналогично шагу #3 производится проверка корректности признаков и их сохранение в БД.

6. **RESULT** В результате мы получаем таблицу, где содержится вся информация, которая потребуется нам для дальнейшего анализа.

Извлечение текста из HTML

Для извлечения текста из загруженных HTML документов были опробованы два подхода: автоматические методы очистки HTML от информационного шума [14] и метод на основе XPath и регулярных выражений, схожий с тем, который применялся для поиска ссылок в разделе 3.1. В результате был выбран второй подход, поскольку первый требовал значительно больше времени для интеграции и настройки. К тому же для в данной работе использовались только два новостных сайта и реализация специального решения способного точно извлекать необходимые данные из страниц с данных сайтов было более оправданно с точки зрения затраченного времени.

3.3. Набор данных

Реализованная система для сбора данных была запущенна на 3 компьютерах, при этом 2 из них (серверы приложений) осуществляли сбор и предобработку данных; третий использовался в качестве сервера РСУБД (PostgreSQL) и сервера СУБД Redis (используется системой для хранения и распределения задач). Автоматизированный сбор данных выполнялся в течении 24 часов, при этом на каждом из компьютеров было задействовано 32 потока для загрузки данных из Интернета, и 3 процесса для задач *парсинга*. Система использует потоки как более легковесную альтернативу процессам ОС для задач не требующих активных вычислений (как правило IO) и процессы в задачах, где требуется больше вычислительных ресурсов. Такое решение было принято из-за особенностей реализации сборщика мусора в интерпретаторе языка Python (CPython 2.7): в один и тот же момент времени физически может выполняться только один поток (в отличии от процессов).

Число 32 потока было получено опытным путем, при этом использовалась возможность динамического выделения ресурсов (см. Рис. 3.3) и возможность просмотра логов в реализованной системе. При большем количестве соединений новостные сайты как правило обрывали большинство соединений.

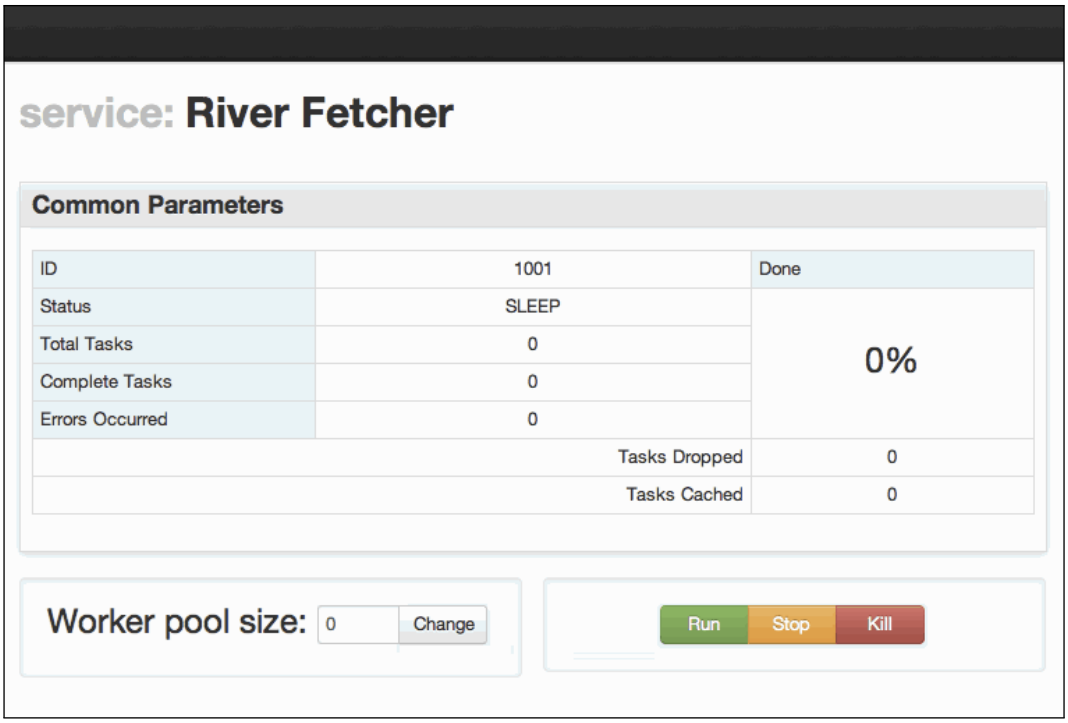


Рис. 3.3: Фрагмент интерфейса системы сбора данных.

Диаграммы UML, подробно описывающие архитектуру системы сбора данных, а также дополнительные скриншоты интерфейса расположены соответственно в приложениях *«В»* и *«Г»* к данной работе.

Используя реализованную систему, было собрано приблизительно 200,000 страниц новостей из новостных сайтов как Forbes.com (50,000 новостей опубликованных с января 2010 по по март 2012) и Businessinsider.com (140,000 новостей опубликованных с января 2009 по по март 2012). Выбор именно этих сайтов обусловлен тем, что они размещают данные о количестве просмотров новостей в открытом доступе, эти данные использовались в качестве показателя популярности новостей. Также, для каждой новости были собранны реакции из социальной сети *Twitter*. Ниже представлены графики иллюстрирующие собранный набор данных: количество новостей (stories), суммарное количество просмотров и реакций в Twitter за день.

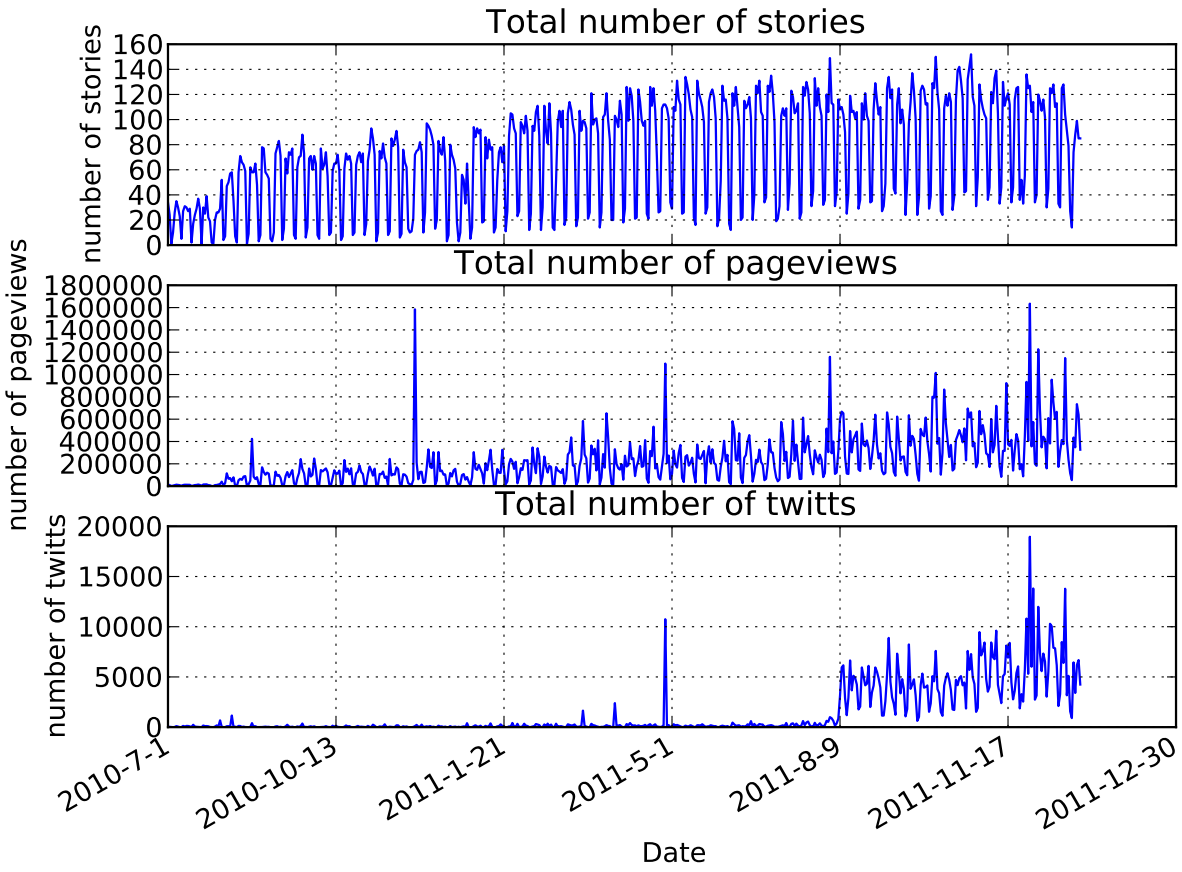


Рис. 3.4: Показатели собранного набора данных Forbes.

4. Предварительная обработка

Как уже упоминалось выше, необходимость этого шага определяется тем обстоятельством, что все методы регрессии, которые будут подробнее рассмотрены в следующем разделе, требуют, чтобы входные данные были представлены в виде последовательностей чисел одинакового размера и одинакового формата. Описываемые ниже методы могут быть использованы в комбинации друг с другом. Чтобы сделать ссылки на методы более удобными, присвоим этим методам короткие буквенные идентификаторы, которыми будем помечать их описание.

4.1. Методы представления текстов в виде векторов признаков

Во всех рассматриваемых методах каждая позиция в векторе признаков соответствует некоторому объекту, наличествующему (или отсутствующему) в тексте. Разные методы различаются типами объектов, соответствующих позициям в векторе, способами вычисления стоящих в векторе значений и способами уменьшения размеров этих векторов.

Первый выбор, который приходится делать при определении процедуры векторизации, это выбор типов объектов, соответствующих позициям в векторе признаков. Здесь возможны следующие варианты.

- **WF**. Каждая позиция соответствует некоторой форме некоторого слова. Например, словоформам «культура» и «культурами» будут соответствовать разные позиции.
- **NF**. Каждая позиция соответствует нормальной форме некоторого слова. Например, словоформам «музейная» и «музейным» будет соответствовать одна позиция, а словам «музей» и «музейному» — разные.
- **SYN**. В этом подходе для группирования слов с близкими значениями в рамках одной позиции вектора признаков (и тем самым уменьшения его размеренности) используется тезаурус. Для целей данного исследования был применен широко известный тезаурус WordNet. Основной структурной единицей этого тезауруса является синсет — синонимическая группа слов. Синсеты связаны отношением гипернимии (отношением «общее — частное»). Каждая позиция в векторе соответствует синсету с учетом отношения гипернимии, так что если, например, в тексте присутствует слово «Windows», то это повлияет на позиции вектора, соответствующие синсетам «операционные системы» и «программы».
- **NE**. Подход, в котором позициям в векторе соответствуют слова и словосочетания являющиеся именованными сущностями (например, названия фирм, имена людей), автоматически находимые в текстах.

Далее идут методы ограничения размерности векторов признаков, которые могут использоваться в комбинации.

- **STOP**. Использование стоплиста, списка частых неспецифичных слов, не несущих информации о смысле текста, как, например, слова «каждый», «вид», «являться». В вектор признаков не включаются позиции, соответствующие словам из этого списка.

- **NOUN**. Используются только существительные.

- **FREQ**. На основе информации об априорных вероятностях встречаемости слов, полученной на большом корпусе текстов, для включения в вектор признаков отбираются только те, чья частота в каком-либо из текстов выходит за пределы (точнее, превышает) границу доверительного интервала, вычисляемого из предположения о равномерном распределении соответствующего объекта (словоформы, слова или синсета) в текстах.

- **LL**. Используются только те признаки, которые имеют ненулеывые коэффициенты линейной модели построенной по алгоритму LASSO [12].

- **SVD**. Используются только те признаки, которые имеют наибольшие сингулярные числа после сингулярного разложения матрицы признаков X^l.

- **COR**. Используются только те признаки, которые имеют коэффициент корреляции с ответами больше некоторого заданного порога.

В итоге была выбрана комбинация **NF+NE+STOP+FREQ+LL**. Методы **WF** и **NOUN** были отклонены так как согласно статье [15] могут приводить к значительной потере информативности векторной модели. Метод **SYN**, требует значительно более сложной процедуры анализа текста и поэтому также не был применен. Методы **SVD** и **COR** требовали уточнения параметра порога, точная настройка которого выходила за рамки данной работы.

Для вычисления весов, стоящих в векторе признаков использовался упомянутый во введении метод TFIDF. Заключительная манипуляция, которая производится с векторами признаков, — это их нормирование: компоненты всех ненулевых векторов делятся на евклидову длину вектора.

4.2. Векторизация текстов

Каждая статья представлена как совокупность признаков: *заголовок*, *содержание*, *дата публикации*, *количество просмотров* и *количество ссылок* на данную новость в социальной сети Twitter. Далее в этом разделе будет описан процесс обработки текстовых признаков, которые для краткости будем называть просто *текст*. Нужно отметить, что обработка заголовка и полного текста новостей осуществлялась отдельно, но одинаковыми методами и в конце раздела будет отдельно упомянуто как результаты этих обработок сводились к одному общему (для отдельной новости) вектору признаков.

Сначала тексты были подвергнуты процедуре *сегментации* [10, сс. 112-116], т.е. разделению текста на отдельные *токены* – лингвистические единицы. В данном случае в качестве токенов выступали слова и пунктуация, также на этом этапе слова не подвергались никакому преобразованию, поскольку это могло бы привести к потере эффективности других алгоритмов при дальнейшей обработке текстов [15, сс. 1-47]. Для реализации сегментации использовался алгоритм *TreeBank* из библиотеки NLTK.

Следующим шагом стала процедура определения частей речи в последовательностях токенов (*part-of-speech tagging, POS*). Библиотека *NLTK* содержит довольно широкий набор алгоритмов (таггеров) для POS-Tagging. Тем не менее, вопрос о том, каким алгоритмом следует воспользоваться должен решаться конечным пользователем библиотеки, исходя из данных которые требуется обработать. Таггер это - классификатор, который отображает пару *«токен–контекст»* во множество POS-классов (меток из расширенного множества частей речи). Процесс построения таггера выглядит следующим образом: сначала алгоритму на вход подается тренировочная последовательность токенов для которых заранее известны целевые классы. Алгоритм “запоминает” какие классы чаще всего встречаются для разных контекстов в тренировочном наборе. Затем, когда на вход алгоритму подается новый контекст для которого метка класса не известна — алгоритм возвращает для этого контекста класс с самой большой априорной вероятностью (частотой) среди тренировочных данных. Что такое контекст, каждый алгоритм POS-Tagging определяет по-своему.

В ходе работы был протестирован следующий ряд алгоритмов и их комбинаций:

- N-граммные таггеры – это таггеры, которые определяют в качестве контекста последовательность из N классифицированных токенов стоящих перед текущим токеном. NLTK содержит несколько реализаций: *Unigramm* (**U**), *Bigramm* (**B**), *Trigramm* (**T**) для которых число N равно 1, 2 и 3 соответственно.

- Морфемные таггеры — определяют в качестве контекста часть токена. NLTK содержит две реализации данного подхода: *Affix* (**A**) (контекст — первые и последние *N* символов токена), *Regexp*(**R**) (контекст — регулярное выражение, которому соответствует токен).

- Таггер на основе *правил трансформации Брилла* (**Br**). Этот таггер отличается от предыдущих алгоритмов тем, что ему требуется другой таггер для начальной классификации к результатам работы которого затем будут применяться правила трансформации Брилла. Суть данного подхода в том что что алгоритм анализирует ситуации в которых ошибается начальный таггер и синтезирует правила коррекции для улучшения качества классификации.

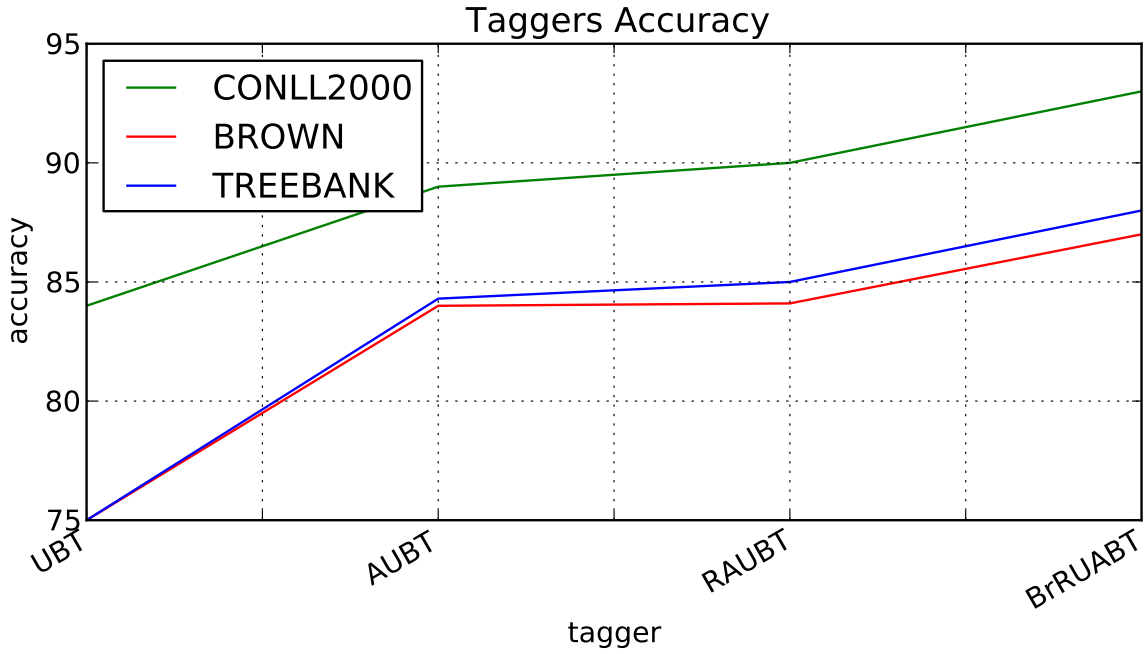


Рис. 4.1: Качество различных алгоритмов POS-Tagging

Таггеры были протестиваны по отдельности (кроме последнего), а также в комбинациях (механизм комбинирования — *backoff chain*). Чтобы настроить и проверить эффективность алгоритмов были использованы следующие общеизвестные коллекции текстов: *Brown.Reviews*, *Conll2002*, *TreeBank* — тематически близкие к Интернет-новостям. Для данных коллекций процедура POS-Tagging была осуществлена вручную авторами коллекций. Результат ручной классификации принимался за эталон (gold), качество (accuracy) алгоритмов POS-Tagging определялась путем сравнения результата его работы с эталоном.

Как видно из графика на Рис.4.1, алгоритм **BrRAUBT** на всех использованных коллекциях текстов обеспечил наибольшее качество классификации, поэтому он был выбран в качестве алгоритма определения частей речи в данной работе.

К последовательностям классифицированных токенов был применен алгоритм распознавания именованных сущностей (*named entity recognition*, NER), которые затем были использованы как отдельные признаки в векторной модели. Этот шаг является важным, поскольку он позволяет частично уменьшить потерю информации при переходе от текста к векторной модели. Векторная модель является относительно простой и одной из самых распространенных и хорошо изученных, а различные алгоритмы работающие с ней значительно быстрее чем алгоритмы работающие с более сложными моделями (например графовыми). Для реализации распознавания использовался алгоритм по-умолчанию из NLTK, который строил для последовательности классифицированных токенов семантическое дерево.

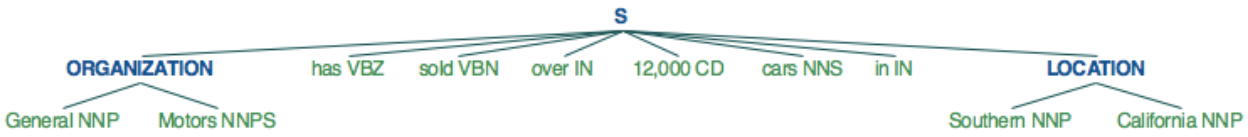


Рис. 4.2: Семантическое дерево — результат работы алгоритма NER.

Рисунок 4.2 иллюстрирует что алгоритм NER распознал название организации (General Motors) и места (Southern California) в приведенном тексте *“General Motors has sold over 12,000 cars in Southern California.”*. Полученное дерево преобразуется в последовательность *термов*. Каждый терм это кортеж: *«токен - POS_класс - NE_класс»*. NE-класс это — класс именованной сущности: место (location), человек (person), организация (organization) или учреждение (facility).

4.3. Отбор значимых признаков

После преобразования текстов в последовательности термов, термы пересчитываюся и составляется частотный словарь, где для каждого терма хранится его *документная частота (DF)* — количество текстов (новостей), в которых встречается этот терм. Используя данный словарь для каждого текста вычисляется вектор *TFIDF*, но поскольку размер полученного словаря достаточно велик (приблизительно 650,000 термов), то вычисление всех документов заняло бы слишком много времени и полученные вектора занимали ли бы слишком большое количество памяти (порядка 500 Gb для собранного набора данных, при условии что каждая позиция вектора занимает 4 байта), то на этом этапе были применены методы ограничения размеренности, рассмотренные в предыдущем подразделе. После применения методов **STOP** и **FREQ** размер словаря удалось уменьшить до 20 тысяч термов. При этом в методе **FREQ** верхний порог документной частоты был $1/2$ и нижний $1/5000$.

Следующей стадией было вычисление TFIDF для уменьшенного словаря. Для каждой новости были вычислены вектора TFIDF текста новости и заголовка, которые затем суммировались с коэффициентами 0.3 и 0.7 соответственно. В итоге, для каждой новости получился один вектор TFIDF, который затем нормировался. Отдельного исследования о том какие коэффициенты стоит выбрать не проводилось, данные числа были взяты из [1], где они рекомендуются в качестве начальных. Полученные таким образом вектора составляют строки матрицы признаков X^l.

Далее к векторам TFIDF добавлялись компоненты, характеризующие дату публикации соответствующей новости: день (значение самой ранней опубликованной новости было установлено на отметке 0), месяц, год, день недели, а также суммарное количество просмотров всех новостей на данном сайте за прошлую неделю. После этого, заключительным этапом было построение линейной модели с использованием метода LASSO:

$$\hat{Y} = \hat{\beta}_0 + \sum_{j=1}^{p} X_j \hat{\beta}_j; \quad \hat{Y} = X^T \hat{\beta} \tag{4.1}$$

Здесь $\hat{\beta}_j$ — коэффициенты линейной модели, $\hat{Y}$ — свободная переменная. Поиск коэфициэтов $\hat{\beta}_j$ производится через минимизацию функционала *Лассо Тибширани*:

$$\hat{\beta}^{lasso} = \arg\min_{\hat{\beta}} \Big\{ \frac{1}{2} \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{i=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \Big\} \tag{4.2}$$

В итоге из модели были удаленны признаки, которые получили нулевые коэффициенты $\hat{\beta}^{lasso}$. Таким образом были получены вектора размерности 177, которые будут использоваться следующем разделе в качестве входных данных для алгоритмов регрессии. Некоторые из отобранных признаков представленны в таблице ниже ((полная таблица отобранных термов в приложении *«Д»*)):

Таблица 4.1: Фрагмент отобранных методом LASSO признаков.

#	терм	коэф.	#	терм	коэф.	#	терм	коэф.
1	psychologists	249673.2612	31	mexican	29538.34565	146	gold	-383.573800
2	happiest	184883.9999	32	meditation	28929.20034	147	fargo+wells	-427.595047
3	microsystems+sun	176956.9915	33	dropbox	28857.38889	148	men	-456.051925
4	tournaments	171402.6753	34	beauty	28185.36552	149	future	-457.035970
5	noise	152565.3813	35	canada	26188.06359	150	forbes	-551.298340
6	surprises	112310.885	36	debit	24316.59606	151	news+video	-761.128138
7	activision	111356.3362	37	headline	23672.12009	152	ios	-902.204183
8	non-existent	93413.50076	38	colleges	23356.76088	153	ceo	-923.208263
9	agents	86653.62601	39	iphone	23340.94358	154	digital	-965.969492
10	contacts	83484.08162	40	analysis	22643.82359	155	etfs	-1481.814784
11	cold	80426.81861	41	bureaucracies	20985.33097	156	trefis	-1579.699442
12	cart	74384.05367	42	atm	20075.59753	157	yahoo	-1748.541857
13	title	71176.65102	43	no	19788.71848	158	influence	-1758.152929
14	teachers	70979.60884	44	public	19247.13694	159	data	-1812.095491
15	dick	68086.02897	45	hundreds	19053.90653	160	etf	-1898.469245
16	world	59608.88955	46	gates	18691.90035	161	david	-2024.087826
17	profession	59069.04908	47	players	18598.10381	162	groupon	-2045.091146
18	porn	57703.2544	48	motor	17613.23659	163	fbi	-2076.593913
19	engineers	45487.97521	49	siri	17240.28694	164	experience	-2299.384806
20	mubarak	44120.81341	50	juniper	17163.35881	165	leaders	-2443.093796
21	sean	39162.37013	51	mind	16974.01633	166	layoff	-2659.585071
22	list	37092.22059	52	funds	16074.07232	167	opportunity	-2763.305678
23	icloud	35252.72888	53	price	15511.65371	168	consumers	-2783.263811
24	pictures	35105.72807	54	camera	15481.10794	169	stewart	-2847.251814
25	tool	33531.59584	55	america	14600.481	170	china	-2992.569917
26	hhs	31997.79213	56	bank	14147.23372	171	uk	-3463.399315
27	linkedin	31470.8517	57	soros	13785.74421	172	workers	-3634.145566
28	power	30125.07202	58	fear	12966.94836	173	career	-4633.149046
29	chase	29710.95103	59	wells	12440.73059	174	colleagues	-4945.640593
30	drawing	29636.82777	60	purchases	12252.70716	175	engineering	-6380.158517

5. Прогнозирование популярности новостей

Цель этого шага — построение алгоритма на основе набора векторов признаков и соответствующих им значений популярности.

Методы регрессии

Существует неизвестная целевая зависимость $y^* : X \to Y$, значения которой известны только на объектах обучающей выборки $X^l = (x_i, y_i)_i^l = 1, y_i = y^*(x_i)$. Требуется построить алгоритм, который в данной задаче принято называть «функцией регрессии» $a : X \to Y$, аппроксимирующий целевую зависимость y^*. Далее рассмотрим модели регрессии которые были использованны в данной работе.

- **KNN**. Алгоритм «ближайших соседей» (*nearest neighbors*). Этот алгоритм делает прогноз на основе *N* обучающих векторов, ближайших к новому вектору для которого значение независимой переменной неизвестно. В качестве прогнозируемого значения зависимой переменной среднее арифметическое зависимых переменных *N* ближайших векторов. В качестве расстояния используется евклидово расстояние в некотором подпространстве векторов признаков. Набор измерений, составляющих это подпространство, параметр *N*, а также некоторые другие параметры алгоритма часто подбираются некоторой оптимизационной процедурой, минимизирующей общую ошибку предсказания на обучающем наборе. Существуют варианты, также учитывающие расстояния до ближайших соседей, но в этой роботе использовался классический подход.

- **ERF** (extreme randomized forrest of trees) — «чрезвычайно случайный лес». Разновидность метода **RF** (случайный лес). В методе **RF** классификатор строится в виде большого количества деревьев решений (decision trees), причем в качестве предсказанного для данного вектора значения свободной переменной выбирается среднее арифметическое значений предсказанных деревьями леса. Деревья, составляющие классификатор, строятся независимо, так, что при расщеплении всех узлов всех деревьев используется лишь небольшое и каждый раз случайно выбираемое подмножество признаков, составляющих вектора. В результате того, что прогноз делается достаточно большим ансамблем независимых классификаторов, достигается статистическая

надежность прогноза даже в случае, когда размерность векторов признаков превышает количество обучающих примеров. В **ERF** как и в методе **RF**, используются случайные подмножества признаков кандидатов, но вместо того чтобы искать пороги для разделяющих правил, они генерируюся случайно для каждого компонента и лучшие из случайно сгенерированных порогов выбираются в качестве разделяющих правил. Как правило, это приводит к уменьшению дисперсии алгоритма, за счет незначительного увеличения смещения.

- **RIDGE**. Ридж-регрессия это — разновидность многомерной линейной регрессии, отличие которой состоит во введении ограничения на норму вектора коэффициентов модели, что приводит к уменьшению некоторых коэффициентов. Метод также приводит к повышению устойчивости модели в случае большого числа обусловленности матрицы признаков X^l, что позволяет получить интерпретируемые модели — отбираются признаки, оказывающие наибольшее влияние на вектор ответов:

$$\hat{Y} = \hat{\beta}_0 + \sum_{j=1}^{p} X_j^l \hat{\beta}_j; \quad \hat{Y} = (X^l)^T \hat{\beta} - \textit{линейная модель} \tag{5.1}$$

$$\hat{\beta}^{ridge} = argmin_{\hat{\beta}} \Big\{ \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{i=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \Big\} \tag{5.2}$$

Настройка моделей проводилась с использованием метода скользящего контроля (K-fold cross validation) с числом сверток равным 10, при этом в качестве настраиваемых параметров в методе **KNN** выступало число соседей K, в методе **RIDGE** — коэффициент λ и в методе **ERF** — критерий качества расщепления (коэффициент Джини или значение информационной энтропии), а также и количество деревьев.

6. Эмпирическое сравнение методов регресии

Исходный набор данных был разделен, путем группировки векторов в непересекающиеся наборы меньшего размера на основе информации о дате публикации соответствующей новости. В один поднабор попадали только новости собранные в течении двух месяцев, начиная с некоторой даты (для каждого набора начальная дата выбиралась на 2 месяца позже чем для предыдущего). На полученных таким образом наборах тестировалась эффективность регрессионных моделей.

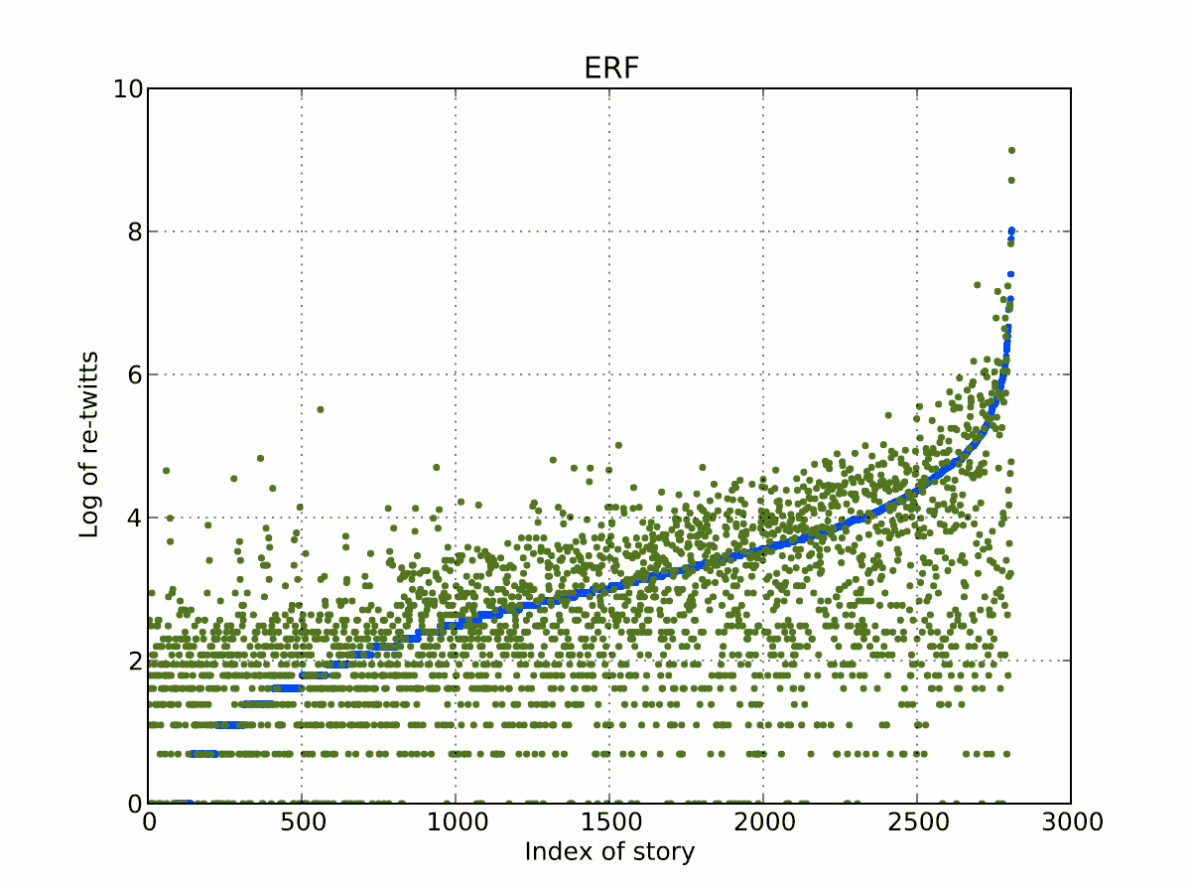
Для тестирования эффективности различных алгоритмов регрессии также использовался метод скользящего контроля: фиксировалось некоторое множество разбиений исходной выборки на две подвыборки: обучающую и контрольную. Для каждого разбиения выполняется обучение алгоритма по обучающей подвыборке с использованием значений параметров полученных в предыдущем разделе, затем оценивалось его средне квадратичная ошибка алгоритма (*MSE*) на объектах контрольной подвыборки. Оценкой скользящего контроля называется средняя по всем разбиениям величина ошибки на контрольных подвыборках. Предполагалось, что выборка независима, следовательно средняя ошибка скользящего контроля давала несмещённую оценку вероятности ошибки.

Наилучшие результаты дал метод **ERF** с параметром количеством деревьев равным равным 50 и коэффициентом Джини в качестве критерия расщепления.

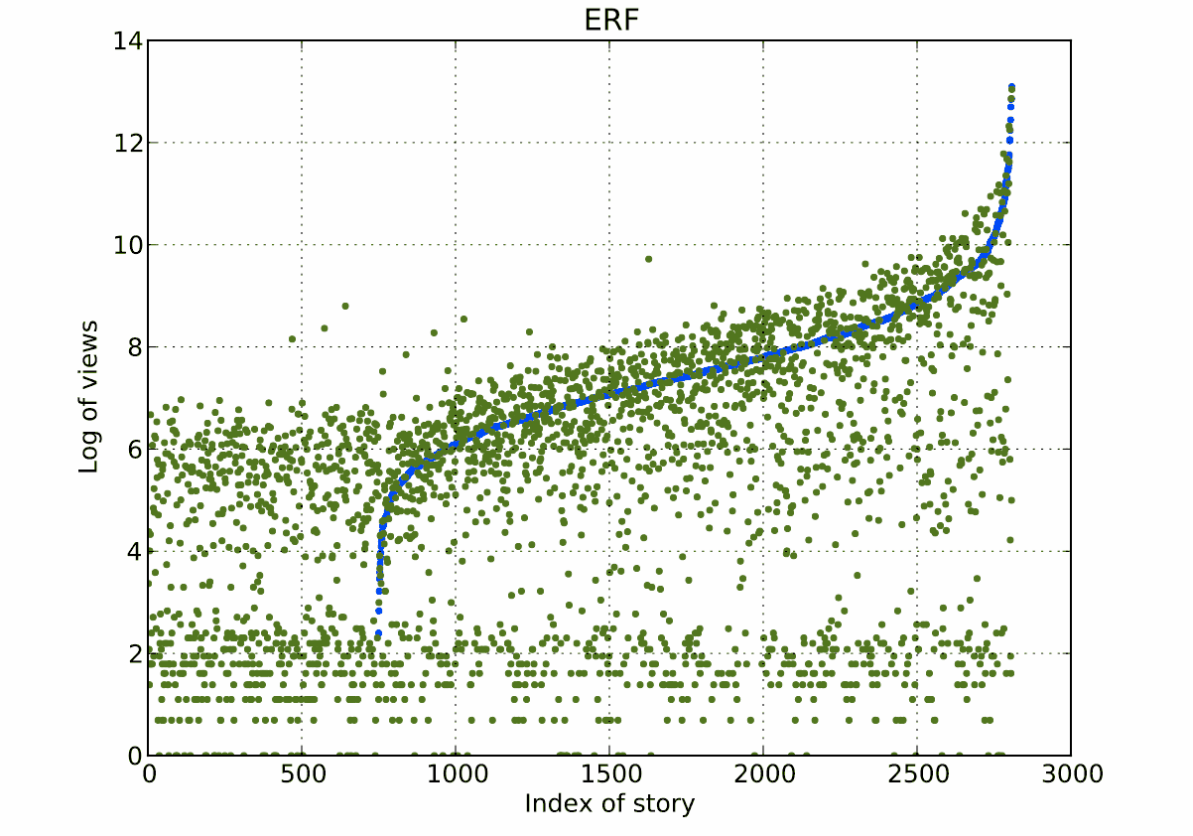
Таблица 6.1: результаты тестирования алгоритмов регрессии

Алгоритм	MSE (re-twits)	MSE (views)
KNN	17396	613959
RIDGE	11306	336794
ERF	9489	26313

Далее на графиках представлены результаты регрессии методом ERF (графики для остальных методов находятся в *приложении «E»*). Везде на графиках синие точки — действительный значения популярности, зеленые — спрогнозированные. Результаты получены на наборе Forbes (2011.09-2011.10). Новости изначально были упорядочены по возрастанию количества просмотров (для большей наглядности), каждой новости был присвоен индекс (ось X).



(a) Прогнозирование количества “ретвитов”



(b) Прогнозирование количества просмотров

Рис. 6.1: Результаты прогнозирования методом ERF

Заключение

Была спроектирована и реализована система обеспечивающая автоматизацию процесса прогнозирования популярности интернет-новостей. Система включает в себя инструменты, необходимые для сбора исходных данных, преобразования текстов в векторную модель с использованием алгоритмов обработки естественных языков. Собираемые данные предоставляются потребителю в удобном структурированном виде, совместимом с пакетом *scikit-learn*, и могут использоваться для прогнозирования.

Система была протестированна на реальных источниках — сайтах Forbes.com и Businessinsider.com, суммарный объем собранных данных был равен $\approx 50Gb$ (200,000 документов HTML). Используя собранные данные, были созданы частотные словари и построена векторная модель документов которая затем оптимизировалась и далее использовалась для прогнозирования. Стоит отметить, что на всех этапах использовались только собранные системой данные. Единственным исключением был словарь стоп-слов (≈ 500 слов), взятый автором из открытого источника: http://www.ranks.nl/resources/stopwords.html.

Заключительным этапом было тестирование алгоритмов машинного обучения для решения задачи прогнозирования популярности собранных новостей. На этом этапе были проверенны методы: Extremely Randomized Forest, K Nearest Neightbours и Linear Ridge Regression. Все методы дали достаточно большую среднеквадратическую ошибку, при этом, учитывая графики в разделе **6.1** и приложении «E» метод **ERF** дает относительно хорошие результаты прогнизирования.

7 Способов улучшить процесс прогнозирования

1. Процесс сбора данных может быть улучшен, если использовать унифицированные по структуре **RSS** (или **Atom**) каналы в качестве источника исходных данных. Такой подход не использовался в данной работе, поскольку требовалось собрать однородный набор данных охватывающий большой временной промежуток, что невозможно было сделать через каналы **RSS** на выбранных новостных сайтов, поскольку они предоставляют доступ только к последним

опубликованным материалам. Как показало тестирование и в силу специфики предметной области, сколь-нибудь точного и быстрого прогнозирования можно добиться используя модель построенную на “свежих” данных. Поэтому в реальном приложении использование **RSS** более чем оправданно.

2. Полностью перейти на метрики популярности контента, основанные на откликах в социальных сервисах (Twitter, Facebook, VK, LinkedIn, Digg, ReadIt и т.д.). Эти сервисы предоставляют доступ к данным в структурированном формате (XML иди JSON), что значительно упрощает задачу реализации и поддержки инструментов сбора данных (в данном случае парсеров). Значительная часть времени реализации данной работы была потрачена иммено на исследование структуры источников, написание парсеров и инструментов для их тестирования. Ошибка на данном этапе может привести сбору не тех данных которые требуется и фатально сказаться на качестве построенных моделей.

3. Использовать больше источников. Это позволит лучше учитывать контекст в котором публикуется контекст. Недостаток контекста предположительно стал причиной недостаточного качества прогнозирования (для реальной системы), поэтому использование большего количества информации об окружении в котором публикуется новость может стать ключем к построению более точных, пригодных к реальным условиям моделей.

4. Использовать дополнительные признаки в модели: информацию об авторе, категорию текста, тональность текста и комментариев, учитывать ссылочную структуру HTML и т.д. Комбинация разных моделей, например, стохастическое моделирование социальной динамики из [4] позволит точнее находить потенциально популярные новости по мере поступления данных от пользователей.

5. Улучшить исспользование машинного обучения, протестировать больше моделей. Возможно более оправданным будет перейти от регрессии к классификации, где классами документа будут выступать метки популярности: «не популярен», «низкая популярность», «высокая популярность» и т.д. Исспользование других методов настройки и отбора значимых признаков также может дать положительный результат.

6. Значительную часть времени заняла процедура частично ручной оптимизации векторной модели и ручной настройки регресиионых моделей. Для сокращения времени на обучения моделей можно использовать потоковые (*online*) реализации алгоритмов, которые обучаются по мере поступления новых данных. Как можно большая автоматизация этих процессов позволит более эффективно тратить время аналитика.

7. При реализации, многие архитектурные решения были приняты в условиях недостатка времени и ресурсов (все время над проектом работал 1 человек) — это негативно сказалось на производительности и архитектуре некоторых частей системы. Например язык Python использовался на всех этапах реализации, чтобы сократить издержки на разработку интерфейсов для взаимодействия между разными компонентами системы. Реализация приведенных выше изменений потребует значительного улучшения архитектуры системы для обеспечения большей автоматизации и производительности. Одним из наиболее мощных инструментов построения распределенных систем на сегодняшний день является язык *Erlang / OTP* — возможно, он станет лучшей альтернативой при реализации каркаса распределенной системы (сейчас для этого используется не самый подходящий, но гораздо более простой фреймворк Celery). Также стоит обратить внимание на развивающийся Open-Source инструмент для распределенного машинного обучения — Apache Mahout. На данный момент, для хранения постоянных данных система использует PostgreSQL и это негативно сказывается на производительности при работе с данными векторной модели, где предполагается использование больших объемов бинарных массивов, что плохо вписывается в реляционную модель. Использование разных решений для разных задач позволит более эффективно использовать имеющиеся ресурсы.

Список литературы

[1] Manning C., Raghavan P., Schutze H. Introduction to Information Retrieval. — Cambridge: Cambridge University Press, 2008. — 585 с.

[2] Hastie T., Tibshirani R., Friedman J. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. — Springer. Stanford, 2009. — 763 с.

[3] Воронцов К. Математические методы обучения по прецедентам: теория обучения машин. — Курс лекций ВМК МГУ и МФТИ, 2011. — 141 с.

[4] Lerman K., Hogg T. Using a Model of Social Dynamics to Predict Popularity of News. // arXiv.org — Cornell University Library, 2010. — http://arxiv.org/abs/1004.5354

[5] Bollen J., Mao H., Zeng X.J. Twitter mood predicts the stock market. // arXiv.org – Cornell University Library, 2010. — http://arxiv.org/abs/1010.3003

[6] Zhang W., Skiena S. Trading Strategies to Exploit Blog and News Sentiment. // Department of Computer Science, Stony Brook University, 2010. — http://www.aaai.org/ocs/index.php/ICWSM/ICWSM10/paper/download

[7] Frisbee B., Ghosh I. The Predictive Power of Financial Blogs. // Haverford College. Dept. of Economics, 2010. — http://hdl.handle.net/10066/4888

[8] Schumaker R. An Analysis of Verbs in Financial News Articles and their Impact on Stock Price. // Iona College, 2010. — http://dl.acm.org/citation.cfm?id=1860669

[9] Lifshits Y. Ediscope: Social Analytics for Online News. // Yahoo — Yahoo Labs, 2010. — http://research.yahoo.com/files/YL-2010-008.pdf

[10] Bird S., Klein E., Loper E. Natural Language Processing with Python. — O'Reilly Media, 2009. — 504 с.

[11] Perkins J. Python Text Processing with NLTK 2.0 Cookbook. — Packt Publishing, 2010. — 625 с.

[12] El Ghaoui L., Viallon V., Rabbani T.
Safe Feature Elimination in Sparse Supervised Learning. // arXiv.org – Cornell University Library, 2010. — http://arxiv.org/abs/1009.3515

[13] Dean J., Ghemawa S.
MapReduce: Simplified Data Processing on Large Clusters. // Google, Inc, 2004. — http://static.usenix.org/event/osdi04/tech/full_papers/dean/dean.pdf

[14] Kohlschutter C., Fankhauser P., Nejdl W.
Boilerplate Detection using Shallow Text Features. // WSDM 2010, 2010. — http://www.l3s.de/kohlschuetter/publications/wsdm187-kohlschuetter.pdf

[15] S. Fabrizio.
Machine Learning in Automated Text Categorization. // ACM computing surveys (CSUR), 2002. — http://nmis.isti.cnr.it/sebastiani/Publications/ACMCS02.pdf