1. Установления взаимосвязей между новостями и твитами

Задача автоматического установления связей между твитами и новостями решена посредством написания программного комплекса, который обладает следующими возможностями:

1. сбор необходимой для решения задачи информации;
2. построение наборов данных;
3. применение к наборам данных методов машинного обучения;
4. получение рекомендаций новостей для произвольных твитов;
5. вариативность в выборе метода для построения рекомендаций;
6. возможность получить информацию о качестве используемого метода.

Программный комплекс реализован с использованием языка программирования Python версии 2.7.

1.1. Архитектура

где-то в главе упомянуть промежуточное хранилище
вступление

Визуализация структуры построенной системы производится при помощи блок-схем [?]. Для удобства восприятия блоки действия (изображаются прямоугольником) выделяются зелёным цветом, а прочие используемые блоки, такие как ввод-вывод данных (изображаются параллелограммом) и хранимые данные (изображаются фигурой, представляющей собой прямоугольник, в котором две противолежащие стороны заменены на две одинаковые и параллельные кривые, совпадающие с секцией окружности), выделяются синим цветом.

Получение данных заключается в скачивании новостей из RSS потоков и твитов, с использованием Twitter Streaming API, в течение длительного промежутка времени, с последующим помещением всех данных в промежуточное хранилище. В работе в качестве хранилища выступает python shelve. Получение данных в виде блок-схемы изображено на рисунке 1.

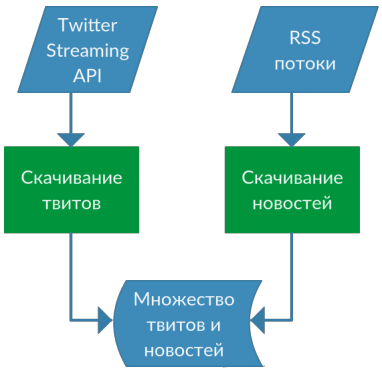

Рисунок 1 — Блок-схема получения данных

На основе полученного множества новостей и твитов происходит автоматическое построение набора данных. Набор данных эта структура состоящая из списка новостей и списка твитов, где для каждого твита указана ссылка на единственную новость.

Результатом работы всех реализованных методов является сопоставление численных векторов (векторов для сравнения) каждому обрабатываемому тексту, с помощью которых можно оценить насколько похожи любые два текста.

Метод TF-IDF не имеет стадии обучения модели, поэтому применяется непосредственно к набору данных и получает вектора для сравнения, для всех текстов, которые были переданы ему на вход. Получаемые вектора обладают размерностью совпадающей с размером корпуса.

В отличие от метода TF-IDF методы WTMF и WTMF-G состоят из двух стадий: обучения и применения модели. На стадии обучения методы строят модель (в сериализованной модели помимо самой модели содержится набор данных, на основе которого была построена модель) и получают вектора для сравнения для всех элементов набора данных. На стадии применения методы WTMF и WTMF-G на основе ранее построенной модели для произвольного множества твитов строят векторы для сравнения полученных на вход твитов и новостей из набора данных.

На основе множества, состоящего из твитов и новостей, для каждого элемента в котором сопоставлен вектор для сравнения, строятся рекомендации. Рекомендации представляют собой множество твитов, к каждому из которых сопоставлен ранжированный по мере убывания схожести список новостей.

На основе построенных рекомендаций можно как произвести оценку качества ранее использованного метода, так и получить их в виде текстового файла, который содержит информацию в пригодном для чтения формате. Оценка качества полученного метода происходит возможно, только если рекомендации были получены из набора данных.

Процесс оценки качества различных методов рекомендаций, а также получение рекомендаций для твитов из набора данных изображён на рисунке 2.

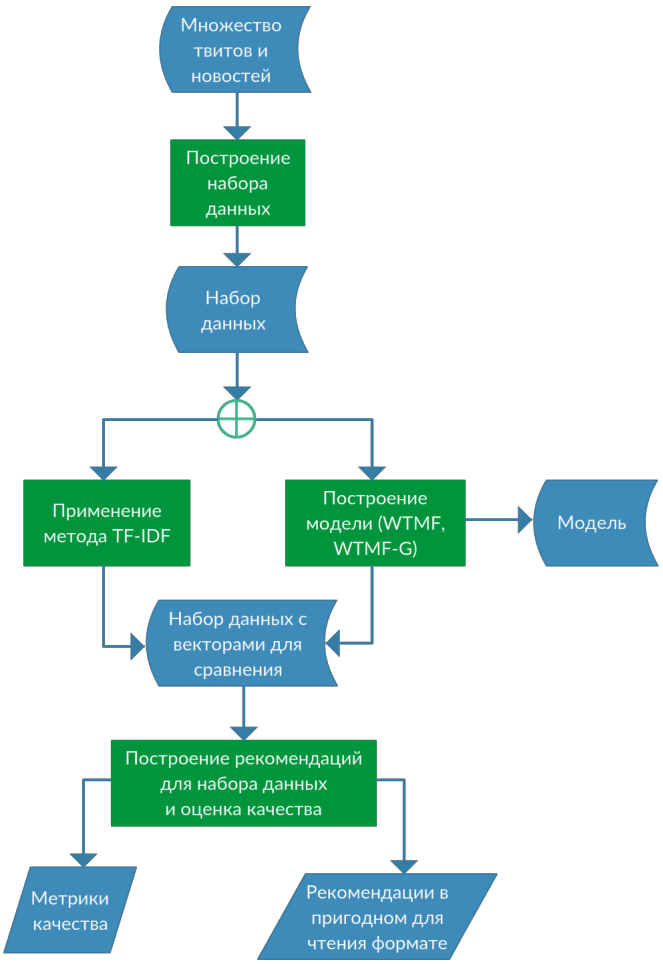

Рисунок 2 — Блок-схема процесса оценки качества используемых методов

Дополнительным результатом изображённого на рисунке 2 процесса является построенная модель (для методов WTMF и WTMF-G), которую можно применить на произвольное множество твитов. Процесс получения рекомендаций для произвольных твитов изображён на рисунке 3.

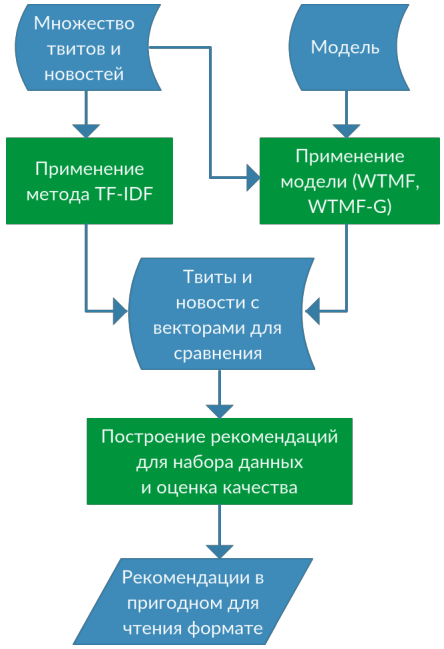

Рисунок 3 — recommend

заключительный абзац

1.2. Обработка естественного языка

Работа посвящена поиску семантической близости текстов, поэтому в ней имеет место использование решений таких задач обработки естественного языка, как:

1. токенизация — разбиение предложения на слова;
2. лемматизация — процесс приведения словоформы к лемме;
3. извлечение именованных сущностей.

Описанные выше задачи решены с использованием набора сторонних библиотек для языка Python, а именно:

1. ntlk — платформа, для написания приложений на языке Python, обрабатывающих естественный язык;
2. pymorphy2 — морфологический анализатор;
3. polyglot — библиотека, позволяющая извлекать именованные сущности из текстов на разных языках.

Для решения задачи токенезации используется стандартный токенизатор, реализованный в ntlk. Задача лемматизации решается в случае русского языка с помощью морфологического анализатора pymorphy2, в случае английского языка с помощью морфологического анализатора WordNet, реализованного в ntlk.

Извлечение именованных сущностей происходит с помощью библиотеки polyglot. В используемой библиотеке реализуется выявление именованных сущностей на основе заранее сформированного и размеченного корпуса именованных сущностей. Корпус формируется на основе данных из Википедии.

1.3. Метод WTMF

Модель для метода WTMF построена на основе мнзаранее подготовленного набора данных. В контексте работы набор данных состоит из множества новостей и твитов, из которых в процессе работы извлекается набор текстов (для твита — текст твита, для новости — конкатенация заголовка и краткого изложения статьи).

По множеству текстов, которые получены из набора данных, построена модель, пригодная для сериализации, состоящая из матрицы P (здесь и далее используются обозначения введённые в главе **??**). Построение модели зависит от четырёх констант:

1. K — размерность вектора, по которому производится сравнение (если TF-IDF матрица X была размера $M \times N$, то по завершении работы алгоритма будут получены две матрицы P размера $K \times M$ и Q размера $K \times N$);

2. I — число итераций алгоритма построения модели;

3. w_M — коэффициент, задающий вес негативного сигнала при построении матрицы весов W;

4. λ — регуляризирующий член.

Применение полученной модели на множество твитов представляет собой следующий процесс: сначала строится TF-IDF матрица X для новостей из набора данных и множества твитов, затем на основе новой матрицы X строится весовая матрица W, и наконец на основе построенных матриц X и W и посчитанной на этапе обучения матрицы P выполняется половина итерации алгоритма обучения, а именно получение матрицы Q по матрице P:

$$Q_{\cdot,j} = (PW'_jP^T + \lambda I)^{-1}PW'_jX_{j,\cdot}.$$

В результате получаем вектора для сравнения твитов из заданного множества.

1.4. Метод WTMF-G

Построение модели для метода WTMF-G основывается на построение модели метода WTMF. Набор данных состоит из множества новостей и твитов и связей вида текст-текст, из которых, в процессе работы извлекается набор текстов. (для твита — текст твита, для новости — конкатенация заголовка и краткого изложения статьи).

По множеству текстов, которые получены из набора данных, построена пригодная для сериализации модель, представляющая собой матрицу P. Построение модели зависит от четырёх констант:

1. K — размерность вектора, по которому производится сравнение (если TF-IDF матрица X была размера $M \times N$, то по завершении работы алгоритма будут получены две матрицы P размера $K \times M$ и Q размера $K \times N$);

2. I — число итераций алгоритма построения модели;

3. w_M — коэффициент, задающий вес негативного сигнала при построении матрицы весов W;

4. δ — коэффициент, задающий степень влияния связей вида текст-текст.

Применение полученной модели на множество твитов производится аналогично применению модели для метода WTMF за исключением двух моментов: во-первых, необходимо на

основе новостей из набора данных и множества твитов перестроить связи текст-текст, во-вторых получение матрицы Q происходит по следующей формуле:

$$Q_{\cdot,j} = (PW'_jP^T + \lambda I + \delta L_j^2 Q_{\cdot,n(j)} diag(L^2_{n(j)}) Q^T_{\cdot,n(j)})^{-1} (PW'_j X_{j,\cdot} + \delta L_j Q_{\cdot,n(j)} L_{n(j)}).$$

В результате получаем вектора для сравнения твитов из заданного множества.

1.5. Эффективная работа с матрицами

Построение и применение моделей WTMF и WTMF-G требует большого количества операций над матрицами, что на практике занимает продолжительное время. Поэтому актуальна задача по повышению эффективности работы с матрицами.

Для эффективной работы с матрицами используются программные библиотеки для языка Python numpy и scipy (базируется на библиотеке numpy и расширяет её функционал).

Повышение производительности при работе с матрицами производится на примере оптимизации времени расчёта формулы получения строк матрицы P, которая используется при построении моделей WTMF и WTMF-G. На каждой итерации построения модели происходит многократное выполнение формулы (число выполнений порядка 10^4, зависит от размера корпуса):

$$P_{i,\cdot} = (QW'_iQ^T + \lambda I)^{-1} QW'_i X^T_{i,\cdot}.$$

В начале была написана наивная реализация алгоритма, которая показала производительность, не приемлемую в рамках решения задачи. Затем наивная реализация оптимизировалась следующим образом:

1. переход к перемножению матриц с использованием высокопроизводительной библиотеки для языка C OpenBlass (в библиотеке numpy существует возможность перейти к использованию для работы с матрицами некоторых библиотек, написанных на языке C [**?**]);

2. сохранение в отдельной переменной переиспользуемых результатов вычислений над матрицами;

3. переписывание кода для работы с разреженными матрицами;

4. удаление лишних приведений матриц к формату python list и обратно.

Результаты оптимизации приведены в таблице 1.

Получили, что оптимизированное решение работает в 325 раз быстрее наивной реализации. Дальнейшая оптимизация не производилась, так как получено решение работающее за приемлемое время.

Таблица 1: Оптимизация работы с матрицами

Добавленная оптимизация	**Время за 100 итераций (с)**	**Прирост производительности (раз)**
Наивная реализация	205	1
Перемножение с помощью OpenBlass	55	3.73
Переиспользование результатов	15.15	3.63
Работа с разреженными матрицами	0.75	20.2
Сокращение количества приведений типов	0.63	1.21