# Содержание

# 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 использование языка программирования Python версии 2.7.

Ниже приводится описание архитектуры программного комплекса, а также разбор отдельных моментов.

## 1.1. Архитектура

где-то в главе упомянуть промежуточное хранилище

вступление

Реализованный программный комплекс выполняют следующие набор функций (мб стоит переписать):

1. получение данных из твиттера;
2. получение данных из новостной rss-ленты;
3. расшифровка коротких URL;
4. автоматическое построение набора данных;
5. построение набора данных на основе вручную размеченного набора твитов;
6. построение моделей для методов WTMF и WTMF-G;

7. построение рекомендаций для методов WTMF, WTMF-G и поиска схожести на основе частнотности употребления слов (TF-IDF);

8. оценка качества рекомендаций;

9. получение результатов рекомендаций в пригодном для чтения формате;

два слова про то что рисуем, рисуем блок-схемами
описание блок схем, согласно такому-то госту.
получение данных, заключается в том-то том-то изображено на рисунке 1


Рисунок 1 — twnews consumer

абзац про построение наборов данных
абзац про векторы для сравнений
абзац про получение моделей и векторов для сравнений датасетов
абзац про метрики и рекомендации
то-то изображено на рисунке 2
рекомендации для произвольных твитов строятся так-то
то-то изображено на рисунке 3
абзац про завершение

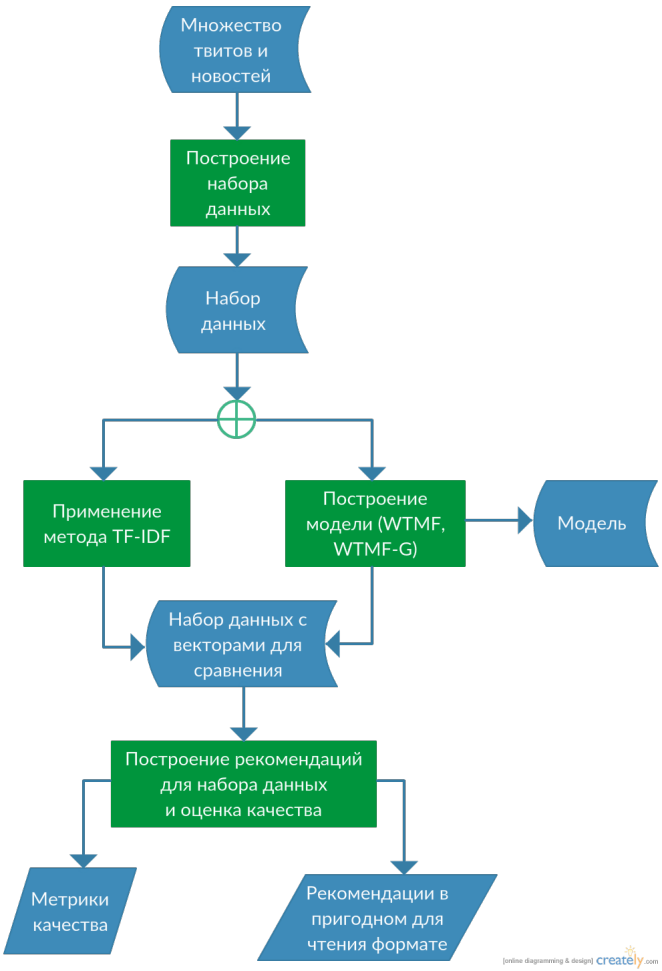


Рисунок 2 — eval

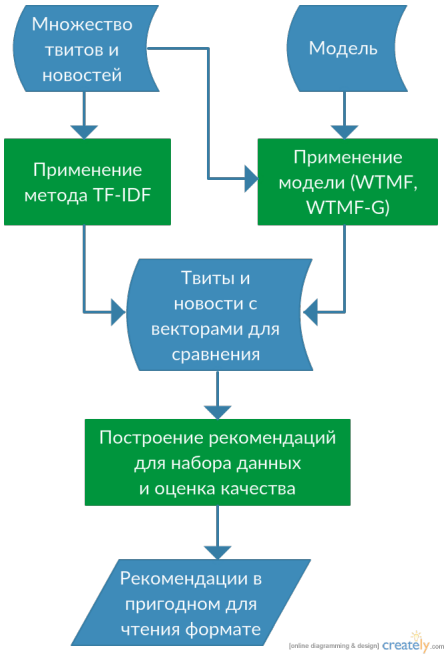


Рисунок 3 — recommend

## 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. $\lambda$ — 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. $\delta$ — 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:

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

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):

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

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.

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 |

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