Описание модуля TextFromHTMLExtractor

**Назначение**: «Вытаскивать» с новостных веб-страниц часть, непосредственно содержащую новостную статью.

Python 2.7

**Описание директории**:

Подпапки:

1. resources – текстовые файлы, необходимые для работы алгоритма
2. results – список проверенных url и файлы с результатами (названия соответствуют номеру ссылки в urls.txt)

Файлы:

1. \_\_init\_\_.py
2. settings.py – файл настроек программы
3. url\_handler.py – обработчик url, поданной на вход.
4. html\_parser.py – парсер полученного html-кода.
5. text\_processor.py – обработчик «сырого» текста
6. article\_extractor.py – файл, в котором вычисляется часть страницы со статьёй
7. output\_formatter.py – файл, в котором происходит форматирование полученной статьи.
8. main.py – исполняемый модуль программы, получающий на вход url из командной строки.

**Сторонние библиотеки**:

Использованы две сторонние библиотеки, не противоречащие первому требованию задания:

1. lxml – для построения дерева разбора html-страницы.
2. pymorphy2 – для лемматизации слов.

**Описание алгоритма**

Тексты на естественном языке содержат большое количество стоп-слов (предлогов, союзов, частиц, местоимений), которые не несут смысловой нагрузки. Но их большая концентрация в одном месте указывает, что возможно это текст. Таким образом, нужно определить эти стоп-слова и составить их список.

В документе-кандидате на звание новостной статьи подсчитывается количество таких стоп-слов (сигнальных слов), отбрасываются все остальные слова. Строится вектор документа-кандидата, вектор списка сигнальных слов. Между векторами вычисляется расстояние на основе [косинусного коэффициента](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D1%8D%D1%84%D1%84%D0%B8%D1%86%D0%B8%D0%B5%D0%BD%D1%82_%D0%9E%D1%82%D0%B8%D0%B0%D0%B8). Чем ближе итоговое значение к 1, тем более похожим считается вектор документа-кандидата на вектор сигнальных слов. Документ, вектор которого ближе всего к 1 выбирается как искомая новостная статья.

Пошагово:

1. Получаем на вход url из командной строки, возвращаем исходный код веб-страницы как юникодную строку.
2. Передаем полученную строку html\_parser'у и получаем дерево разбора.
   1. Выделяем часть <body></body>, т.к. именно в ней содержится статья.
   2. Удаляем узлы типа <script>, <style>, <noindex>, <footer> и пр. как нерелевантные для поиска в них статьи.
   3. В оставшемся теле <body> ищем и извлекаем теги-параграфы (<p>, <pre>, <blockquote>, <code>, <cite>), т.к. именно в них содержатся части статьи.
   4. Для этих параграфов находим родителей, таким образом, мы собираем множество параграфов в один родительский узел.
   5. Поскольку lxml оперирует своими структурами etree.Element., переводим получившиеся узлы в юникод для текстовой обработки
   6. Здесь же получаем заголовок статьи по тегу <title>
3. Переходим в text\_processor, который проводит разбиение на слова и лемматизацию в полученных родительских узлах с параграфами. Возвращает список списков сигнальных слов.
4. Далее модуль article\_extractor для каждого текста считает частотность сигнальных слов, преобразует список частот в вектор и вычисляет косинусный коэффициент. Тексты сортируются по убыванию значения коэффициента и выбирается первый текст, как содержащий тело новостной статьи.
5. Модуль output\_formatter берет найденную статью, которая заключена в теги и очищает её от тегов. Если находит ссылки, то заключает url в квадратные скобки. Строки разбиваются на подстроки блиной 80 символов.
6. Модуль main объединяет все другие модули и сохраняет статью в текстовый файл.

**Направления дальнейшего улучшения/развития программы**:

1. Отладить форматирование, т.к. не везде остается по одной пустой строке между абзацами.
2. Делать проверку валидности url. Сейчас, если подать программе невалидную ссылку, она упадет.
3. Сделать многопотоковую/асинхронную загрузку страниц для того, чтобы обрабатывать ссылки из заданного массива.
4. Расширить файл настроек – сейчас в нём можно задать только язык и списки стопслов и сигнальных слов.
5. Если на странице не найдены параграфы (теги <p>, <pre>, <blockquotes> и пр.) возвращаться к дереву и искать по другим элементам, например по <span>.
6. Добавить ещё языки для иностранных сайтов, составить для них списки сигнальных слов.