САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

КАФЕДРА ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

Альбрант Сергей Евгеньевич

Выпускная квалификационная работа бакалавра

Автоматическое восстановление хронологии событий по новостным публикациям на русском языке

Направление 010400

Прикладная математика и информатика

Заведующий кафедрой,
кандидат технических наук,
доцент Блеканов И. С.

Научный руководитель,
ст. преподаватель Малинина М. А.

Рецензент,
кандидат физико-математических наук,
доцент Лепихин Т.А.

Санкт-Петербург

2017

Содержание

Введение

В жизни современного человека интернет играет огромную роль. Основная причина этого заключается в информативности упомянутого ресурса. С помощью интернета люди могут быстро получать информацию о последних событиях во всем мире.

Во всемирной паутине каждый день появляется неисчислимое количество новостных публикаций, при этом все большее количество людей используют именно интернет для ознакомления с новостями.

Новостные публикации часто содержат в себе описания всевозможных событий, которые происходят в различное время. Некоторые последовательные и относящиеся друг к другу события могут быть указаны в различных новостных статьях. Зачастую, людям интересно проследить хронологию событий, узнать, как они развивались, что и когда происходило или произойдет.

Решение задачи, поставленной в данной работе, поможет выделить среди многообразия новостных публикаций события с соответствующими им датами и упорядочить их хронологически.

Постановка задачи

Целью данной работы является разработка программы, способной из некоторой новостной коллекции на русском языке выделить *темпоральные выражения* вместе с соответствующими им событиями и отсортировать их в порядке хронологии.

Темпоральным выражением будем называть последовательность символов (слова, цифры и знаки), которая обозначает время, выраженное моментом времени, периодом или частотой.

Для достижения поставленной цели необходимо реализовать следующие задачи:

- Формирование обучающей и тестовой коллекции новостных публикаций.
- Разделение полученных текстов новостей на предложения, а предложений на слова.
- Разметка обучающей коллекции с точки зрения наличия
- Составление лингвистических правил для применения методов машинного обучения;
- Определение наличия темпорального выражения в каждом предложении с использованием методов машинного обучения.
- Приведение темпоральных выражений к единому формату и упорядочивание их в хронологическом порядке вместе с соответствующими им событиями.

Обзор литературы

Задача извлечения темпоральных выражений уже давно стала вызывать интерес. Еще с начала 80-х годов прошлого века проводятся международные конференции, выявляющие в процессе соревнования лучшие результаты обработки текстов на темпоральные выражения, сравнивая их с эталонной разметкой, сделанной экспертами.

Наиболее известным достижением является язык разметки TimeML [1], первая версия которого была представлена в 2004 году. TimeML — язык разметки временных выражений и упоминаний событий в естественном языке. Он позволяет размечать: упоминания событий и их связь с темпоральными выражениями; порядок событий относительно друг друга; нечеткие временные выражения; продолжительность событий;

В 2009 году язык TimeML был признан как международный стандарт для разметки времени и событий в тексте Международной организацией по стандартизации — ISO-TimeML [2].

Но данный язык разметки используется для англоязычных текстов, в то время как для текстов на русском языке его пока применить нельзя. О возможном применении TimeML для русскоязычных текстов рассказано в статье [3].

В отличие от английского языка, для которого уже разработаны достаточное количество мощных и свободно доступных компонент, таких как большое количество разнообразных лингвистических банков данных и аннотированных текстов, для русского языка степень развитости данных компонент довольно ограничена. Также русский язык менее формализован, по сравнению с английским, что усложняет задачу поиска темпоральных выражений в текстах, написанных на русском языке.

Глава 1. Описание применяемых методов и алгоритмов

Основные подходы для нахождения темпоральных выражений можно разделить на три группы:

- методы, основанные на машинном обучении;
- методы, основанные на лингвистических правилах;
- комбинированные методы.

Для решения данной задачи в работе были взяты комбинированные методы, а именно были составлены лингвистические правила, выделяющие основные черты темпоральных выражений. На основе этих правил были применены различные методы машинного обучения и рассмотрена их эффективность для данной задачи. Далее в этой главе рассмотрим математические основы использованных в работе методов машинного обучения.

1.1 Линейная регрессия

Линейная регрессия является методом восстановления зависимости одной переменной от другой.

Рассмотрим линейную функцию:

$$y(x, w) = w_0 + \sum_{j=1}^{p} x_j w_j + \varepsilon = x^T w + \varepsilon, \quad x = (1, x_1, \dots, x_p)$$

Где ε - случайная ошибка модели, распределенная нормально с нулевым математическим ожиданием и фиксированной дисперсией, которая не зависит от переменных x, y.

Пусть дано тренировочное множество $X=\{x_1,\dots,x_l\}$, представим его в виде матрицы:

$$X = \begin{pmatrix} x_{1,0} & \dots & x_{1,p} \\ \vdots & \vdots & \vdots \\ x_{l,0} & \dots & x_{l,p} \end{pmatrix}$$

И задано множество ответов:

$$Y^* = \begin{pmatrix} y_1^* \\ \vdots \\ y_l^* \end{pmatrix}$$

Для того чтобы найти оптимальные параметры $\widehat{w}$ по тренировочным данным X и Y^*, воспользуемся методом наименьших квадратов, минимизировав следующий функционал:

$$Q(w) = \frac{1}{l}\sum_{i=1}^{l}(y_i^* - x_i^T w)^2 = \frac{1}{l}(Y^* - Xw)^T(-Xw)$$

Продифференцировав данное выражение, получим:

$$X^T(Y^* - Xw) = 0$$

Откуда:

$$w = (X^T X)^{-1} X^T Y^*$$

Несмотря на свою простоту и быстродействие, метод линейной регрессии имеет некоторые недостатки:

- неприспособленность к решению существенно нелинейных задач
- чувствительность к большим одиночным выбросам, связанным с грубыми ошибками в данных
- переобучение

1.2 Наивный байесовский классификатор

Наивный байесовский классификатор – это алгоритм классификации, основанный на теореме Байеса с допущением о независимости признаков.

Теорема Байеса позволяет рассчитать апостериорную вероятность P(y|x) согласно следующей формуле:

$$P(y|x_1, \dots x_n) = \frac{P(y)P(x_1, \dots x_n|y)}{P(x_1, \dots x_n)},$$

где $P(y|x_1, \dots x_n)$ – апостериорная вероятность данного класса y (т.е. данного значения целевой переменной) при данном значении признака x.

P(y) – априорная вероятность данного класса.

$P(x_1, \dots x_n|y)$ – правдоподобие (вероятность данного значения признака при данном классе).

$P(x_1, \dots x_n)$ – априорная вероятность данного значения признака.

Используя наивное предположение о независимости, получим:

$$P(y|x_1, \dots x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1, \dots x_n)},$$

Так как $P(x_1, \dots x_n)$ является константой, то ее значение мы можем отбросить. В итоге для классификации получается следующая формула:

$$\hat{y} = \arg\max_{y} P(y)\prod_{i=1}^{n} P(x_i|y),$$

Существует несколько типов моделей, основанных на наивном байесовском алгоритме:

- *Gaussian* (нормальное распределение). Данная модель используется, если признаки непрерывны и их значения имеют нормальное распределение.
- *Multinomial* (мультиномиальное распределение). Данная модель используется, если признаки дискретны.
- *Bernoulli* (распределение Бернулли). Используется в случае бинарных дискретных признаков, то есть принимающих только два значения: 0 и 1.

В дальнейшем для исследования в этой работе будет использован именно третий тип модели – с распределением Бернулли. Для данного типа функция правдоподобия считается следующим образом:

$$P(x_1, \dots x_n|y) = \prod_{i=1}^{n} P(x_i|y)^b (1 - P(x_i|y))^{(1-b)} \quad (b \in 0,1).$$

Наивный байесовский классификатор имеет существенные преимущества:

- Классификация выполняется легко и быстро.

- Если наивное предположение о независимости выполняется, то он превосходит другие алгоритмы, при этом требуя меньший объем обучающих данных.
- Алгоритм лучше работает с категорийными признаками, чем с непрерывными.

Несмотря на свои плюсы, наивный байесовский алгоритм также имеет и недостатки:

- Значения спрогнозированных вероятностей не всегда являются достаточно точными.
- Алгоритм предполагает независимость признаков, но в реальности наборы с полностью независимыми признаками встречаются нечасто.

1.3 Дерево принятия решений

Дерево принятия решений — это средство поддержки принятия решений, использующееся в статистике и анализе данных для прогнозных моделей. Метод деревьев решений является одним из популярных методов решения задач классификации и прогнозирования.

Дерево решений состоит из «ветвей» и «листьев». Ветви дерева хранят в себе значения атрибутов, от которых зависит целевая функция, а листья - значение целевой функции.

Алгоритм построения дерева принятия решения, состоит в том, что вначале необходимо выбрать атрибут и поместить его в корневой узел. Затем, из набора данных для каждого значения i выбранного атрибута (например, бинарный атрибут, со значениями «1» и «0») выбираются только те, для которых значение этого атрибута равно i. Далее, рекурсивно производится построение дерева принятия решений.

Основной вопрос состоит в том, на каком основании определяется каждый следующий атрибут Q? Для этого существует несколько частных

алгоритмов, реализующих дерево принятия решений – основными из которых являются алгоритмы ID3, C4.5 и CART.

В данной работе будет использован алгоритм CART. В его основе лежит использование, так называемого индекса Gini, оценивающего расстояние между распределениями классов.

$$Gini(c) = 1 - \sum_{j} p_j^2,$$

Где c – текущий узел, а p_j – вероятность класса j в узле c.

Среди прочих методов машинного обучения, метод дерева принятия решений имеет ряд достоинств:

- Прост в понимании и интерпретации;
- Не требует подготовки данных;
- Быстрый процесс обучения;
- Позволяет работать с большим объемом информации без специальных подготовительных процедур.

Недостатки метода:

- Незначительные изменения в наборе данных могут приводить к построению совершенно другого дерева. Это связано с иерархичностью дерева;
- Проблема переобучения;
- Проблема получения оптимального дерева решений бывает NP-полной;
- Могут появиться слишком сложные конструкции, которые при этом недостаточно полно представляют данные.

Глава 2. Сбор и обработка данных

2.1 Формирование новостной коллекции

В роли исходных данных в работе выступают новостные публикации на русском языке. В качестве источника этих публикаций был взят новостной портал фонтанка.ру[4], на сайте которого находится общедоступный архив новостных публикаций, начиная с 2000-го года. Для формирования новостной коллекции был написан парсер на языке программирования Python 3.5.

Парсинг реализован на основе библиотеки lxml, предназначенной для обработки XML и HTML страниц. Благодаря данной библиотеке были собраны тексты новостей и даты их публикации, которые могут понадобиться при наличии в тексте неявно выраженных дат.

Для получения приемлемого результата, содержащего в своей хронологии более однородные события, новости были взяты только из одной рубрики - «Политика». В качестве обучающей выборки было взято 114 новостных публикаций, общее количество предложений в которых составляет 1000 предложений. Для тестовой выборки взято 100 новостных публикаций с 1060 предложениями.

2.2 Подготовка данных к обучению

Полученные в результате парсинга новостные публикации необходимо привести к виду, пригодному для применения лингвистических правил и методов машинного обучения.

Для этого имеющиеся публикации первоначально разбиваются на предложения, а затем полученные предложения на слова. Данные разбиения производится с помощью функций sent_tokenize и word_tokenize из специализированной библиотеки для обработки естественного языка NLTK[5]. Также производится дополнительная очистка от лишних символов, таких как литералы или кавычки.

В итоге получается двумерный список, в котором в каждой строке располагаются слова и знаки препинания из каждого предложения, идущие в порядке нахождения в данном предложении. Количество строк в данном списке равно общему числу предложений в тексте публикаций.

Глава 3. Поиск темпоральных выражений

3.1 Разметка обучающей коллекции

Как уже говорилось ранее, для русского языка нет необходимого количества аннотированных текстов для определения темпоральных выражений. Поэтому для решения поставленной задачи пришлось вручную разметить обучающую выборку, состоящую из 1000 предложений. В текстах выделялись темпоральные выражения.

При разметке новостной коллекции учитывались либо выражения, содержащие в себе указанные явным образом даты, либо выражения, с помощью которых мы можем определить явные даты (например, «вчера», «сегодня» и т.д.).

Ниже приведены некоторые примеры предложений и их разметка:

1) *Ранее сообщалось, что глава заполярного поселка Териберка Татьяна Трубилина, в котором проходили съемки фильма «Левиафан», также выступила против проката фильма на больших экранах.*

В данном предложении содержится такое выражение, как «ранее сообщалось», но оно является неявным. По нему, невозможно определить конкретной даты совершения события. Поэтому данное предложение было помечено, как не содержащее темпорального выражения.

2) *6 декабря 2013 года Сердюкову было предъявлено обвинение в халатности, однако следствие было прекращено в связи с амнистией экс-министра.*

Это предложение содержит явно выраженную дату, поэтому оно было размечено положительно.

3) *Сегодня на сайте Интерпола появилось объявление о том, что экс-главу государства разыскивают судебные власти Украины "для уголовного преследования или отбывания наказания".*

Данное предложение содержит слово «сегодня», что позволяет определить конкретную дату, используя дату публикацию новости.

3.2 Составление лингвистических правил

Для решения поставленной задачи были выбраны комбинированные методы поиска темпоральных выражений. Важной частью этих методов является составление лингвистических правил, способствующих определению наличия темпоральных выражений в предложении.

Составленные правила представляют собой бинарные характеристические признаки (принимают значения «0» и «1»), и выглядят они следующим образом:

- Наличие в предложении количественного числительного;
- Наличие в предложении слов из специально созданного словаря, содержащего в себе наиболее характерные для темпоральных выражений слова;
- Наличие в предложении предлога «в»;
- Наличие в предложении порядкового числительного;
- Наличие в предложении цифр;
- Наличие в предложении двоеточия;
- Наличие в предложении точек;
- Наличие в предложении какой-либо цифры и тире, стоящих рядом друг с другом.

Стоит отметить, что в словаре, наличие слов из которого является одним из характеристических признаков, содержатся слова следующих подгрупп:

- Месяца года (январь, февраль...)
- Дни недели (понедельник, вторник…)
- Наречия (вчера, сегодня, завтра, послезавтра, позавчера, сейчас, в настоящее время...)
- Время дня (утро, день, вечер, ночь, полночь)

Для проверки таких характеристических функций, как наличие в предложении порядкового или количественного числительных, а также приведения слов в нормальную форму для сравнения со словарем, была использована библиотека pymorphy2[6].

Pymorphy2 — это морфологический анализатор, разработанный на языке программирования Python и выполняющий лемматизацию и анализ слов.

3.3 Применение классификатора и поиск темпоральных выражений

После формулировки характеристических признаков и разметки обучающей выборки, можно применять методы машинного обучения, рассмотренные в предыдущей главе. В результате получаем классификатор, определяющий наличие темпорального выражения в предложении. Результаты и сравнение рассмотренных методов будут приведены в следующей главе.

Теперь, когда определены предложения, содержащие темпоральные выражения, необходимо выделить даты, указанные в этом предложении, и привести их к единому формату.

Для этого была использована библиотека dateparser[7] для языка Python, позволяющая производить парсинг дат на более чем 20 языках, в том числе на русском. Все найденные даты были приведены к формату «datetime».

Пример работы библиотеки:

```
>>> dateparser.parse(u'13 января 2015 г. в 13:34')
datetime.datetime(2015, 1, 13, 13, 34)
```

Для корректного определения дат сначала из текста выделялись выражения, которые могут являться датами. Далее с помощью парсера данные выражения приводились к формату datetime. Если найденные

выражения не были распознаны парсером, то они не рассматривались. Такими выражениями, например, являлись: «46 тысяч», «9 млн.», «1160 предметов» и т.д.

Помимо полных дат, также рассматривались неполные, такие как:

- *Даты, содержащие только число и месяц (год не указан)*
  В этом случае даты дополнялись до полного формата, на основании года публикации новости. Но иногда год публикации может не соответствовать подразумеваемому в предложении году. Например, предложение *«30 декабря Олега Навального – брата Алексея Навального приговорили к 3,5 годам лишения свободы по «делу об Ив Роше»,* содержащееся в новостной публикации от 14 января 2015 года, говорится о событии, состоявшемся в декабре 2014 года. В данном случае, для правильной работы программы, проверялось наличие глагола в прошедшем времени, а также сравнивались месяц публикации новости и месяц, найденный в предложении.
- *Даты, содержащие месяц и год (день не указан)*
- *Даты, содержащие год (день и месяц не указан)*
  В этих 2 случаях приведение к полной дате не представлялось возможным, поэтому они были отнесены в различные группы дат, образованных на основании полноты дат.

Также были приведены к формату datetime такие выражения, как «сегодня», «вчера» и т.д., исходя из даты новостной публикации.

Все полученные даты были разбиты на три группы, в зависимости от их полноты:

- Полные даты (например, «23 марта 2016 года»)
- Даты, содержащие только месяц и год, без конкретного дня (например, «январь 1994 года»)

- Даты, содержащие только год (Например, в «2016 году»)

Сортировка, восстанавливающая хронологию, производится отдельно в каждой из трех групп.

Глава 4. Результаты и выводы

4.1 Сравнение классификаторов

Рассмотрим результаты различных классификаторов, созданных на основе ранее рассмотренных характеристических признаков и размеченных предложений.

Оценка классификаторов была произведена на основе точности, полноты и f-меры.

Точностью (precision) является доля предложений действительно принадлежащих данному классу относительно всех документов которые система отнесла к этому классу.

Полнота (recall) - это доля найденных классификатором предложений принадлежащих классу относительно всех документов этого класса.

Точность и полнота определяются следующим образом:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

где: TP — количество случаев, когда положительная оценка классификатора совпала с положительной экспертной оценкой;

FP — количество случаев, когда оценка классификатора – положительная, а экспертная оценка - отрицательная;

FN — количество случаев, когда оценка классификатора – отрицательная, а экспертная оценка - положительная;

F-мера представляет собой гармоническое среднее между точностью и полнотой. Рассчитать ее можно по следующей формуле:

$$F = 2\frac{Precision \times Recall}{Precision + Recall}$$

Общее количество темпоральных выражений в обучающей выборке составило 150. Всего была взята выборка из 1000 предложений.

Таблица 1. Сравнение классификаторов

	Линейная регрессия	Наивный байесовский классификатор (Бернулли)	Дерево принятия решений	Экспертная оценка
Найдено предложений, содержащих темпоральные выражения	193	151	141	150
Из них найдено верных	140	128	125	150
Общее количество верно классифицированных предложений	937	955	959	1000
Общее количество неверно классифицированных предложений	63	45	41	0
Точность	0,725	0,848	0,887	-
Полнота	0,933	0,853	0,781	-
f-мера	0,816	0,850	0,831	-

Как видно из таблицы, лучшую точность показал классификатор, использующий алгоритм дерева принятия решений. Но по полноте он уступил двум другим классификаторам. В итоге, наилучший показатель f-меры был получен при использовании наивного байесовского классификатора.

4.2 Итоговая сортировка

Как было сказано ранее, сортировка классифицированных предложений с определенными в них датами была произведена по трем отдельным группам, в зависимости от полноты даты.

Стоит отметить, что лучший итоговый результат можно наблюдать в группе с полными датами. Так как новости публикуются каждый день, а, следовательно, хронологию событий с полной датой проще проследить.

Например, проанализируем новости за январь 2017 года. Рассмотрим кусок работы программы в группе с полными датами, в котором можно легко проследить взаимосвязь событий, расположенных в хронологии недалеко друг от друга:

1) Издание связало это с прошедшей 20 января инаугурацией президента США Дональда Трампа.

Найденная дата: 2017-01-20

2) Срок полномочий Джо Байдена завершится послезавтра, вице-президентом США при Дональде Трампе будет Майк Пенс.

Найденная дата: 2017-01-20

3) Интервью на телеканале будет показано 21 января.

Найденная дата: 2017-01-21

4) Отметим, что отчетно-выборный съезд организации состоится 21-22 января в Москве.

Найденная дата: 2017-01-21

5) Россия каждый день отражает тысячи кибератак со стороны Запада. Об этом официальный представитель Кремля Дмитрий Песков заявил в интервью телеканалу BBC, которое выйдет в рамках программы HardTalk 21 января.

Найденная дата: 2017-01-21

6) Президент США Дональд Трамп раскритиковал в Twitter массовые протесты, прошедшие в США 21 января, на следующий день после его инаугурации.

Найденная дата: 2017-01-21

7) По различным подсчетам, до 2 миллионов человек вышли в субботу, 21 января, на марши протеста против Трампа в США.

Найденная дата:2017-01-21

8) Съезд партии «Единая Россия» проходит в Москве 21-22 января.

Найденная дата: 2017-01-21 00:00:00

Как видно из примера, в рассмотренные дни было довольно много новостей об инаугурации и различных ссылок на нее.

Также стоит отметить, что в 5 предложении из приведенного примера (про кибератаки) было выведено 2 предложения. Это связано тем, что в предложении с датой содержится указательное местоимение, и вся информативная часть содержится в предыдущем предложении. Поэтому все предложения, содержащие в начале сочетание предлога и указательного местоимения «это», были выведены в итоговом результате вместе с предыдущим предложением.

Некоторые предложения, где без контекста не ясен смысл, можно понять, прочитав ближайшие хронологически события. Например, с помощью предложения 8, можно понять о какой организации идет речь в предложении 4, заметив, что эти события одинаковые.

В группах же с неполными датами результаты получились хуже. Во-первых, количество предложений с неполными датами оказалось меньше, чем с полными. Во-вторых, связь между предложениями с неполными датами проследить оказалось труднее. Рассмотрим, например, небольшую выборку из результатов работы программы в группе, где указан только год.

1) Страны ЕС, США и ряд других стран ввели санкции в отношении России в 2014 году, после чего они несколько раз расширялись и продлевались.

Найденная дата: 2014 year

2) Ткачев был включен в расширенный санкционный список Европейского Союза в 2014 году, когда занимал пост губернатора Краснодарского края.

Найденная дата: 2014 year

3) В 2014 году «Нацфронт» получил кредит в € 9 млн в Первом чешско-российском банке (ПЧРБ).

Найденная дата: 2014 year

4) В конце 2014 года этой компании нужно было платить по валютным кредитам, а западные банки денег на перекредитование не дали.

Найденная дата: 2014 year

5) Если же деньги всё-таки были, как в 2014 году, напечатаны, то это может ослабить рубль, но мы этого не заметим, это будет размазано по всему году.

Найденная дата: 2014 year

6) Отметим, что в 2014 году, также в соседнем округе — МО Коломна депутатствует мать депутата Соловьева Нина Киселева.

Найденная дата: 2014 year

Видно, что связи между событиями прослеживаются меньше, так как найденные события является более разбросанными из-за отсутствия конкретного дня и месяца. Но можно проследить связь между 1 и 2 предложениями, которые касаются санкций 2014 года, а также, например, в 5 предложении говорится про ослабление рубля в 2014 году, что тоже связано с указанными санкциями.

Заключение

В ходе написания данной работы была разработана программа, выделяющая из новостной русскоязычной коллекции события с темпоральными выражениями и сортирующая их в порядке хронологии.

Были реализованы следующие модули программного комплекса:

1) Сбор коллекции новостных публикаций.

2) Разделение полученных текстов новостей на предложения, а предложений на слова.

3) Использование методов машинного обучения для определения наличия темпоральных выражений в каждом предложении с помощью произведенной ранее разметки и составленных лингвистических правил.

4) Поиск темпоральных выражений в найденных предложениях.

5) Приведение темпоральных выражений к единому формату и упорядочивание их в хронологическом порядке вместе с соответствующими им событиями.

В дальнейшем планируется улучшить результаты исследования, сделать лингвистические признаки более точными и охватывающими, повысить точность поиска темпоральных выражений.

Список литературы

1. Markup Language for Temporal and Event Expressions http://www.timeml.org/

2. J. Pustejovsky, K. Lee, H. Bunt, L. Romary. “ISO-TimeML: An International Standard for Semantic Annotation”, Proceedings of the 7th International Conference on Language Resources and Evaluation, LREC’10 (Valletta, Malta, 17–23 May, 2010), pp. 394–397.

3. Н. С. Ландо, TimeML для разметки русскоязычных текстов. Оценка перспектив, Программные системы: теория и приложения, 2016, том 7, выпуск 4, 249–265

4. Новости Санкт-Петербурга — Фонтанка.ру http://www.fontanka.ru/

5. Natural Language Toolkit - NLTK 3.0 documentation http://www.nltk.org/

6. Морфологический анализатор pymorphy2 https://pymorphy2.readthedocs.io/en/latest/

7. Dateparser – python parser for human readable dates https://dateparser.readthedocs.io/en/latest/

Приложение

```
import numpy as np
import pickle
import dateparser
import re

months=['январь','февраль','март','апрель','май','июнь','июль','август',
'сентябрь','октябрь','ноябрь','декабрь']
input = open('data_12_04.pkl', 'rb')
data = pickle.load(input)
input.close()

input = open('dates_12_04.pkl', 'rb')
dates = pickle.load(input)
input.close()

input = open('data_train.pkl', 'rb')
X = pickle.load(input)
input.close()

with open('train1.txt') as f:
    y = f.read().splitlines()
y = np.array(list(map(int, y)))
#y = np.loadtxt('train3.txt', dtype=int)
data=data[100:300]
dates=dates[100:300]
print(len(data))
print(len(dates))
```

```
import pymorphy2
from nltk.tokenize import sent_tokenize
from nltk.tokenize import word_tokenize

morph = pymorphy2.MorphAnalyzer()
import re
#f = open('1.txt', 'r', encoding="utf8")
sentences=[]
sentences1=[]
positions=[]
#sentences1= sent_tokenize(f.read())
#f = '\n'.join(data[916:5316])
all_dates = []
#for i in range (1016,1216):
for i in range (len(data)):
    f = ''.join(data[i])
    sentences1.extend(sent_tokenize(f))
#!!!!f = '\n'.join(data[1316:1616])
#f = '\n'.join(data[1216:1416])


f1=open('positions.txt', 'r')
positions=word_tokenize(f1.read())
f1.close()

f2=open('dict.txt', 'r')
dict=word_tokenize(f2.read())
f2.close()
```

```
l=len(sentences1);
k=0;

for i in range (l):
        sentences.extend(re.split(r'\n', sentences1[i]))



sentences = [x for x in sentences if x]
sentences[0] = sentences[0].replace(u'\ufeff', '')

l=len(sentences)

A = [0] * l

for i in range (l):
       A[i]=word_tokenize(sentences[i])

all_dates=[]
for i in range (len(sentences)):
   for j in range (len(dates)):
      if sentences[i] in data[j]:
         all_dates.append(dateparser.parse(dates[j]))
         break

B = []

for i in range(l):
   B.append([])
   for j in range(9):
```

```
        B[i].append(0)

C = []

for i in range(l):
    C.append([])
    for j in range(9):
        C[i].append(0)

for r in range (len(sentences)):

    pos_dict=0
    pos_num=0

    for t in range (len(A[r])):

        if '«' in A[r][t]:
            A[r][t]=A[r][t][1:]

        if '»' in A[r][t]:
            A[r][t]=A[r][t][:-1]

        if 'NUMR' in morph.parse(A[r][t])[0].tag:
            B[r][0]=1
            if C[r][0]!=0:
                C[r][0]=C[r][0]+'; '+A[r][t]+' '+A[r][t+1]
            else: C[r][0]=A[r][t]+' '+A[r][t+1]
```

```
if morph.parse(A[r][t])[0].normal_form in dict and A[r][t]!='лет':
    B[r][1]=1
    #C[r][1]=A[r][t]
    pos_dict=t
    if C[r][1]!=0:
        C[r][1]=C[r][1]+'; '+A[r][t]
    else: C[r][1]=A[r][t]



if 'Anum' in morph.parse(A[r][t])[0].tag:
    B[r][3]=1
    if C[r][3]!=0:
        C[r][3]=C[r][3]+'; '+A[r][t]+' '+A[r][t+1]
    else: C[r][3]=A[r][t]+' '+A[r][t+1]

if any(i.isdigit() for i in A[r][t]):
#if A[r][t].isdigit():
    B[r][4]=1
    pos_num=t
    if C[r][4]!=0:
        C[r][4]=C[r][4]+'; '+A[r][t]+' '+A[r][t+1]
    else: C[r][4]=A[r][t]+' '+A[r][t+1]

if ':' in A[r][t]:
    B[r][5]=1
    if C[r][5]!=0:
        C[r][5]=C[r][5]+'; '+A[r][t-1]+' '+A[r][t]
    else: C[r][5]=A[r][t-1]+' '+A[r][t]

if '.' in A[r][t] and len(A[r][t])>1:
```

```
            B[r][6]=1
            if C[r][6]!=0:
                C[r][6]=C[r][6]+'; '+A[r][t-1]+' '+A[r][t]
            else: C[r][6]=A[r][t-1]+' '+A[r][t]



        if '-' in A[r][t] and A[r][t][0].isdigit():
            B[r][7]=1
            if C[r][7]!=0:
                C[r][7]=C[r][7]+'; '+A[r][t]+' '+A[r][t+1]
            else: C[r][7]=A[r][t]+' '+A[r][t+1]

        if B[r][4]==1 and B[r][1]==1:
            if B[r][8]!=1:
                if abs(pos_dict-pos_num)<4:
                    B[r][8]=1

        if ('в' in A[r][t] or 'B' in A[r][t]) and len(A[r][t])==1:
            B[r][2]=1
            if C[r][2]!=0:
                C[r][2]=C[r][2]+'; '+A[r][t]+' '+A[r][t+1]
            else: C[r][2]=A[r][t]+' '+A[r][t+1]



    #if 'в' in A[r]:
            #B[r][2]=1
x=np.array(B)

from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier()
```

```
y_pred = clf.fit(X, y).predict(x)
y2 = clf.predict_proba(x)
clf.score(X,y)

final_dates = []
periods = []
positions = []

for i in range(l):
    final_dates.append([])
    periods.append([])
    positions.append([])

for r in range (len(sentences)):
    if C[r][4]!=0 and y_pred[r]==1:
        words = C[r][4].split(";")
        t=0
        while t < len(words):
            try:
                if (not
(dateparser.date.DateDataParser().get_date_data(words[t]).get('date_obj') is None))
and len(words[t])>5 and not ('лет' in words[t]) and not (',' in words[t]):
                    if (t+1<len(words)):
                        if (sentences[r].find(words[t])+len(words[t]))-
sentences[r].find(words[t+1])<5: #для 2 фраз
                            if not
(dateparser.date.DateDataParser().get_date_data(words[t]+words[t+1]).get('date_o
bj') is None):
```

```
final_dates[r].append(dateparser.date.DateDataParser().get_date_data(words[t]+words[t+1]).get('date_obj'))

periods[r].append(dateparser.date.DateDataParser().get_date_data(words[t]+words[t+1]).get('period'))
                                positions[r].append(sentences[r].find(words[t]))
                                if (t+2<len(words)):
                                    t+=2
                                    continue
                                else:
                                    break
                        if re.findall(r"[0-9]{4}", words[t])==[]: #для 1 фразы
                            past=0
                            for q in range (len(A[r])):
                                if 'past' in morph.parse(A[r][q])[0].tag and 'VERB' in morph.parse(A[r][q])[0].tag:
                                    past=1
                                    break


                            if all_dates[r].month<dateparser.parse(words[t]).month and past==1:

a=dateparser.date.DateDataParser().get_date_data(words[t]).get('date_obj').replace(year=all_dates[r].year-1)
                            else:

a=dateparser.date.DateDataParser().get_date_data(words[t]).get('date_obj').replace(year=all_dates[r].year)
```

```
                    #print(words[t])
                    final_dates[r].append(a)

periods[r].append(dateparser.date.DateDataParser().get_date_data(words[t]).get('p
eriod'))
                    positions[r].append(sentences[r].find(words[t]))

                if not
(dateparser.date.DateDataParser().get_date_data(word_tokenize(words[t])[0]).get('
date_obj') is None) and re.findall(r"[0-9]{4}", word_tokenize(words[t])[0])!=[]:
                    if morph.parse(A[r][A[r].index(word_tokenize(words[t])[0])-
1])[0].normal_form in months:
                    b=morph.parse(A[r][A[r].index(word_tokenize(words[t])[0])-
1])[0].normal_form+' '+A[r][A[r].index(word_tokenize(words[t])[0])]

final_dates[r].append(dateparser.date.DateDataParser().get_date_data(b).get('date_
obj'))

periods[r].append(dateparser.date.DateDataParser().get_date_data(b).get('period'))
                    positions[r].append(sentences[r].find(words[t]))

print(morph.parse(A[r][A[r].index(word_tokenize(words[t])[0])-
1])[0].normal_form+' '+A[r][A[r].index(word_tokenize(words[t])[0])])
                    else:

final_dates[r].append(dateparser.date.DateDataParser().get_date_data(word_tokeni
ze(words[t])[0]).get('date_obj'))

periods[r].append(dateparser.date.DateDataParser().get_date_data(word_tokenize(
words[t])[0]).get('period'))
```

```
                    positions[r].append(sentences[r].find(words[t]))

                except Exception:
                    if not
(dateparser.date.DateDataParser().get_date_data(word_tokenize(words[t])[0]).get('
date_obj') is None) and re.findall(r"[0-9]{4}", word_tokenize(words[t])[0])!=[]:
                        if morph.parse(A[r][A[r].index(word_tokenize(words[t])[0])-
1])[0].normal_form in months:
                            b=morph.parse(A[r][A[r].index(word_tokenize(words[t])[0])-
1])[0].normal_form+' '+A[r][A[r].index(word_tokenize(words[t])[0])]

final_dates[r].append(dateparser.date.DateDataParser().get_date_data(b).get('date_
obj'))

periods[r].append(dateparser.date.DateDataParser().get_date_data(b).get('period'))
                            positions[r].append(sentences[r].find(words[t]))

print(morph.parse(A[r][A[r].index(word_tokenize(words[t])[0])-
1])[0].normal_form+' '+A[r][A[r].index(word_tokenize(words[t])[0])])
                        else:

final_dates[r].append(dateparser.date.DateDataParser().get_date_data(word_tokeni
ze(words[t])[0]).get('date_obj'))

periods[r].append(dateparser.date.DateDataParser().get_date_data(word_tokenize(
words[t])[0]).get('period'))
                            positions[r].append(sentences[r].find(words[t]))


                t+=1
```

```
numbers=[]
for r in range (1000):
    #if y_pred[r]==1:
    if y_pred[r]==1 and len(final_dates[r])>1:
        print(r+1,' предложение: ')
        print(sentences[r])
        print('Классификатор:',y_pred[r])
        print(max(y2[r]))
        print('Значения признаков: '+str(B[r]))
        if C[r][0]!=0:
            print('Количественные числ-е: '+str(C[r][0]))
        if C[r][1]!=0:
            print('Слово из словаря: '+str(C[r][1]))
        if C[r][2]!=0:
            print('Предлог "в": '+str(C[r][2]))
        if C[r][3]!=0:
            print('Порядковые числ-е: '+str(C[r][3]))
        if C[r][4]!=0:
            print('Числа: '+str(C[r][4]))
            print(str(final_dates[r]))
            print(str(periods[r]))
            print(str(positions[r]))
        if C[r][5]!=0:
            print('Двоеточия: '+str(C[r][5]))
        if C[r][6]!=0:
            print('Точки: '+str(C[r][6]))
        if C[r][7]!=0:
            print('Число-тире: '+str(C[r][7]))
```

```
        print('\n\n')

import datetime
for r in range (len(sentences)):
    for t in range (len(A[r])):
        if final_dates[r]==[]:
            if morph.parse(A[r][t])[0].normal_form=='вчера':
                final_dates[r]=[all_dates[r] - datetime.timedelta(days=1)]
                periods[r]=['day']

            if morph.parse(A[r][t])[0].normal_form=='сегодня':
                final_dates[r]=[all_dates[r]]
                periods[r]=['day']

            if morph.parse(A[r][t])[0].normal_form=='завтра':
                final_dates[r]=[all_dates[r] + datetime.timedelta(days=1)]
                periods[r]=['day']

            if morph.parse(A[r][t])[0].normal_form=='позавчера':
                final_dates[r]=[all_dates[r] - datetime.timedelta(days=2)]
                periods[r]=['day']

            if morph.parse(A[r][t])[0].normal_form=='послезавтра':
                final_dates[r]=[all_dates[r] + datetime.timedelta(days=2)]
                periods[r]=['day']

for_sort=[]

for r in range (len(sentences)):
    check=-1
```

```
        q=0
        zu=0
        s=''
        if (y_pred[r]==1 or periods[r]!=[]):
            for t in range (len(A[r])):
                if 'это'==morph.parse(A[r][t])[0].normal_form and y_pred[r]==1:
                    if t==1:
                        if 'PREP' in morph.parse(A[r][t-1])[0].tag and 'Vpre' in
morph.parse(A[r][t-1])[0].tag:
                            s=sentences[r-1]+' '+sentences[r]
                            for_sort.append([s, final_dates[r][0], periods[r]])
                            break
                if s=='':
                    for_sort.append([sentences[r], final_dates[r][0], periods[r]])



    so1=0
    for r in range (len(final_sort)):
        if 'day' in final_sort[r][2]:
            print(final_sort[r][0])
            print('\nНайденная дата:')
            print(final_sort[r][1].strftime("%Y-%m-%d"))
            print('\n\n\n')
            so1+=1

    so2=0
    for r in range (len(final_sort)):
        if final_sort[r][2][0]=='month':
            print(final_sort[r][0])
```

```
        print('\nНайденная дата:')
        print(final_sort[r][1].strftime("%Y-%m"))
        print('\n\n\n')
        so2+=1

so3=0
for r in range (len(final_sort)):
    if final_sort[r][2][0]=='year':
        print(final_sort[r][0])
        print('\nНайденная дата: '+final_sort[r][1].strftime("%Y year"))
        print('\n')
        so3+=1
```