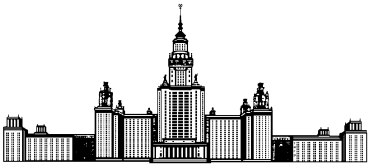
Филиал Московского государственного университета имени М. В. Ломоносова в городе Севастополе



Факультет компьютерной математики
Кафедра программирования

ДИПЛОМНАЯ РАБОТА СТУДЕНТА ГРУППЫ ПМ-501

Извлечение описаний событий предопределённых типов из потока сообщений пользователей микроблогов

Выполнил:
студент 5 курса группы ПМ-501
Е. Ю. Пышнограев

Научный руководитель:
к.ф.-м.н.
Д. Ю. Турдаков

Севастополь, 2014

Аннотация

В работе рассматривается задача извлечения описаний событий из потока сообщений пользователей сети Твиттер. Приведена постановка задачи. Рассмотрены особенности событий в сети Твиттер. Проанализированы существующие подходы к решению задачи извлечения событий. Предложен алгоритм на основе тематической модели HDP с модификацией частичного обучения. Описаны составляющие части алгоритма. Алгоритм реализован, приведен код программы, диаграмма классов и описание компонентов. Рассмотренна работа предложенного алгоритма на тестовых данных, показаны результаты. Найдена точность, полнота и F-мера разработанного алгоритма, показано что на тестовых данных предложенный алгоритм работает лучше, чем модификация существующего алгоритма New Event Detection.

Содержание

Введение

В современном мире информация быстро устаревает, поэтому способы вовремя находить нужные данные — постоянный объект для исследований. Одним из направлений в этой области является извлечение данных из платформ микроблогов.

Платформы микроблогов стали очень популярным способом размещения данных в Сети. В них можно найти сообщения пользователей практически на любую тему, начиная стихийными бедствиями и заканчивая рейтингами музыкальных исполнителей. Правильная обработка доступной информации — нетривиальная задача, которая имеет множество областей применения. Последние несколько лет эта тема активно исследуется во многих университетах мира.

Отслеживание сообщений о стихийных бедствиях в реальном времени поможет вовремя организовать спасательные операции и сохранить жизни людей [1]. Руководствуясь сообщениями пользователей микроблогов можно судить о популярности товаров и вовремя принимать экономически целесообразные решения. Можно делать предположения о рейтингах политических деятелей и эффективности рекламы на основании информации в микроблогах. Помимо перечисленных способов применения доступной информации в микроблоггинговых платформах можно привести множество других.

В данной работе в дальнейшем будет рассматриваться сервис микроблогов Твиттер (http://www.twitter.com). В нем помимо текстовой информации можно публиковать фото, видео и геотеги, что так же может быть использовано при анализе. В доступном наборе данных можно проводить анализ разных сущностей, эта работа посвящена выявлению событий среди потоков информации и извлечению их описания. Поскольку трактовка событий в сообщениях микроблогов может быть субъективной, выделим несколько свойств, которыми характеризуется событие.

Событие в первую очередь является чем-то аномальным на фоне остальных данных. Оно определяется резким изменением частотных характеристик некоторых слов в сообщениях. События в микроблогах носят “взрывной” характер, в течении нескольких часов частоты релевантных слов возрастают в десятки раз и так же быстро опускаются до нормального уровня. Примером события могут быть: стихийное бедствие, выход законопроекта на резонансную тему, получение фильмом награды на кинофестивале.

Существуют разные подходы для решения описанной задачи. В следующих двух разделах будут даны формальные определения и рассмотрены некоторые подходы для решения задачи извлечения событий.

1 Постановка задачи

Цель дипломной работы состоит из нескольких частей:

– исследовать существующие подходы по извлечению описаний событий из сообщений пользователей, выделить возникающие проблемы и рассмотреть возможные методы их решения,
– исследовать возможность применения тематических моделей для решения задачи выявления событий,
– разработать метод для извлечения описаний событий из сообщений пользователей сети Твиттер на основе иерархического процесса Дирихле,
– протестировать работу алгоритма на реальных данных.

Объект изучения этой работы — алгоритм, который по входным данным строит множество событий. В качестве данных для задач подобного рода служит корпус документов (сообщений):

$$\Omega = \left\{ D_i \mid i \in \overline{1, n} \right\}. \tag{1}$$

Будем считать, что каждый документ D_i имеет временную метку t_i. В свою очередь документ D_i определяется как упорядоченный набор слов:

$$D_i = \left\{ w_j \mid j \in \overline{1, l_i} \right\}, \tag{2}$$

при этом слова в документах принадлежат словарю V.

Событие — некоторая сущность, которая характеризуется временем возникновения и ключевыми словами. Оно вызывает резкий подъем частотных характеристик некоторых слов. Событием может быть футбольный матч и музыкальный концерт. В социальной сети Твиттер есть популярные темы, которые всегда генерируют много сообщений. Например это сообщения с ключевыми словами iphone и ipad. Но такие сообщения нельзя считать событиями. Также событиями нельзя считать еженедельные пятничные сообщения о конце рабочей недели [2].

Особенности социальной сети Твиттер состоят в следующем:

– короткие сообщения (до 140 символов),
– наличие шума и ошибок,
– большая плотность сообщений,
– “взрывной” характер событий.

2 Обзор существующих решений рассматриваемой задачи

Прежде чем составлять решение задачи были изучены существующие подходы. Все они комбинируют техники машинного обучения, обработки текстов, вероятностных графических моделей и других разделов науки. Разные методы лучше работают для одних данных и хуже для других, на их качество также влияет природа данных. Выделим несколько подходов, для того чтобы изучить общую схему подобных алгоритмов и чтобы обозначить идеи, которые могут быть использованы при составлении метода извлечения событий из сообщений сети Твиттер. После этого рассмотрим способы решения нескольких подзадач, которые будут входить в разработанный алгоритм.

2.1 New Event Detection

Исторически первым подходом к извлечению событий принято считать NED (New Event Detection) [3]. NED предназначен для того, чтобы находить первый документ на тему, которая не встречалась раньше. Следующие документы на эту тему уже не будут новыми и не будут помечены алгоритмом. Для того, чтобы отвечать на вопрос, является ли документ новым, необходимо указать способ как определять степень сходства двух документов.

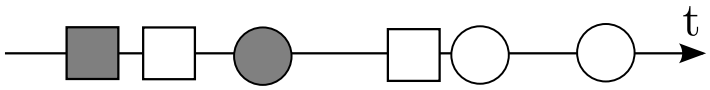


Рисунок 1. Документы, соответствующие двум разным темам. Новые документы помечены серым цветом

Для этого в алгоритме NED используется техника Incremental TF–IDF (Term Frequency – Inverse Document Frequency). TF–IDF — базовый метод выяснять насколько отдельно взятые слова характеризуют весь документ, другими словами, насколько большой вес имеет слово w в документе d. Пусть $f(d, w)$ — количество слов w в документе d. Определим значение $df_t(w)$ как количество документов, поступивших не позднее времени t, в которых встречается слово w. Используя введенные величины, можно записать значение веса определенного слова w в документе d. В момент времени t имеем:

$$\text{weight}_t(d, w) = \frac{1}{Z_t(d)} f(d, w) \cdot \log \frac{N_t}{df_t(w)}, \tag{3}$$

где N_t — общее количество документов, поступивших не позднее времени t, $Z_t(d)$ — нормализационное значение:

$$Z_t(d) = \sqrt{\sum_w \left[f(d, w) \cdot \log \frac{N_t}{df_t(w)} \right]^2}. \tag{4}$$

.

Теперь можно записать значение похожести двух документов, q и d:

$$\text{sim}_t(d, q) = \sum_w \text{weight}(w, d) \cdot \text{weight}(w, q). \tag{5}$$

Указанные формулы записаны в косинусной метрике, также могут быть использованы метрики Хеллингера, Кульбака–Лейблера и другие.

Для того, чтобы понять, является ли добавленный в момент времени t документ q новым, необходимо вычислить степень его похожести со всеми предыдущими документами. Пусть d^* — документ, максимально похожий на q:

$$d^* = \operatorname*{argmax}_d \operatorname{sim}_t(d, q). \tag{6}$$

Тогда значение

$$\operatorname{score}_t(q) = 1 - \operatorname{sim}_t(d^*, q) \tag{7}$$

может быть использовано для того, чтобы определить, является ли документ q новым. Новыми будем считать все документы q, у которых значение $\operatorname{score}_t(q)$ больше, чем пороговое значение θ_s. В обратном случае считается что существует документ d^*, достаточно похожий на q и поэтому q не является сообщением на новую тему. Для того, чтобы определить подходящее значение θ_s, можно использовать размеченный корпус и посчитать значение $\operatorname{sim}_t(d, q)$ среди документов, соответствующих одному и разным событиям.

Недостатком алгоритма NED является то, что он не учитывает отношения между словами. Например синонимичные слова в NED будут считаться совершенно разными. Для того, чтобы решить эту проблему могут быть применены тематические модели.

2.2 Выявление событий с помощью LDA

На следующем примере рассмотрим как тематические модели (topic models) могут быть использованы для задачи распознавания событий. Для этого опишем схему, по которой работает алгоритм, предложенный в статье [4].

Задача состоит в том, чтобы по данным к набору фотографий в сети Flickr распознать все события на определенную тему, проходившие в обозначенных городах. Есть несколько вариантов постановок задачи, они различаются тематикой событий и городами.

Фотографии содержат в себе описания, они будут являться документами в задаче распознования. Также фотографии могут включать геотеги. Авторы статьи предлагают разбить решение задачи на пять составных частей:

1. предобработка текстовых данных,
2. определение в каком городе сделана фотография по ее описанию,
3. распознавание темы, к которой относится фотография,
4. распознавание события, к которому относится фотография,
5. оптимизация описания события, полученного на шаге 4.

В качестве предобработки, авторы предлагают выполнить следующее: удалить стоп-слова и html теги, провести стемминг слов[1], перевести не английские слова на английский язык используя сервис Google Translate.

Так как задача упоминает отдельные города, необходимо разработать метод распознавания города по документу. Указанные в фотографии географические координаты были доступны авторам в 20% случаев, из этих координат были выявлены города используя сервис Google Tables. Используя технику TF–IDF, в описаниях фотографий с известными городами были извлечены ключевые слова. По этим ключевым

[1]стемминг (stemming) — удаление окончаний у слов для их нормализации

словам появилась возможность назначить фотографиям без геотегов “ближайший” город с точки зрения похожести описаний. Для того чтобы определять похожесть документов, был использован подход, описанный в (2.1). В случаях, когда “ближайший” город выявить не удавалось, авторами использовались следующие предположения: считалось что один и тот же фотограф не мог побывать в один день более чем в двух разных городах, и что путешествие из одного города в другой занимает как минимум два часа. Эти предположения позволили улучшить классификатор, таким образом более 97% фотографий были привязаны к городу.

Далее необходимо для каждого города кластеризовать документы по темам и рассмотреть только те из них, которые заданы в описании задачи. Для распознования тем была использована тематическая модель LDA (Latent Dirichlet Allocation) [5], для определения параметров которой применялось сэмплирование по Гиббсу [6]. LDA работает из предположения, что каждый документ D_i характеризуется случайным распределением над темами, в то время как каждая тема является мультиномиальным распределением над словами.

Оставшаяся часть — извлечение событий и их оптимизация. Для того, чтобы алгоритм выявил событие, отвечающее теме k в день d, необходимо чтобы количество документов D_i по этой теме в день d превосходило некоторое пороговое значение θ. Оптимизация событий подразумевает под собой объединение событий на одну тему в последовательные дни и разделение событий в разных городах.

Авторы статьи тестировали алгоритм на трех разных вариантах условия задачи, в таблице 1 приведены результаты по каждому из них.

Таблица 1. Точность, полнота и F-мера алгоритма при разных условиях задачи

Данные	Точность	Полнота	F-мера
1	80.98	19.25	31.10
2	91.21	77.85	84.00
3	90.76	81.91	86.11

По результатам из таблицы можно видеть что в первом случае алгоритм справляется со своей задачей существенно хуже, чем в других. Авторы объясняют это тем, что в задаче 1 необходимо было находить научные конференции, а так как они проходят на разные темы, не получалось выделить конкретный набор ключевых слов. По этой причине полнота алгоритма в случае 1 относительно низкая. В задаче 2, 3 напротив удалось обозначить необходимые ключевые слова, о чем свидетельствуют результаты.

2.3 Information nuggets

Рассмотрим подход к извлечению событий, описанный в [1]. Авторы ставят перед собой задачу составить алгоритм описания подсобытий в социальной сети Твиттер. Были использованы данные, полученные во время торнадо Joplin в 2011 году. Данные представляют из себя сообщения пользователей, содержащие хэштег #joplin, собранные 22 мая 2011 года на протяжении нескольких часов, пока плотность сообщений не стала относительно низкой. Авторы ставили цель извлекать из потока сообщений так называемые золотые самородки информации — короткие и информативные сообщения, описывающие происходящие события. По этой причине подход назван information nuggets.

Авторы видели основную проблему в том, что даже при наличии большого количества сообщений об одном событии, ими трудно пользоваться, потому что они имеют разную природу. Например это может быть сообщение очевидца о происходящем стихийном бедствии или сообщение о перечислении правительством средств на восстановление разрушенных построек. Статья предлагает делить сообщения на пять разных категорий по степени информативности:

- Персональное: информация в сообщении может быть полезна только автору и его кругу общения. Она не является интересной для людей, которые не знают автора сообщения непосредственно.
- Информативное (напрямую): сообщение может быть полезно людям вне круга общения автора, и эта информация написана прямым участником или очевидцем событий.
- Информативное (косвенно): сообщение может быть полезно людям вне круга общения автора, при этом автор пишет о том, что он слышал по телевидению, радио или любому другому источнику информации.
- Информативное (напрямую или косвенно): сообщение может быть полезно людям вне круга общения автора, но невозможно ответить на вопрос как автор связан с происходящими событиями.
- Другое: сообщение или не на английском языке, или не может быть классифицировано.

В дальнейшем рассматриваются только сообщения с информативным типом, так как они наиболее вероятно содержат полезные сведения. Затем сообщения разбиваются на подтипы по направленности информации. Всего авторами использовалось 32 разных подтипа, среди которых:

- Предостережения и советы: документ содержит предупреждение о возможном опасном происшествии.
- Жертвы и разрушения: в тексте сообщается о потерях, вызванных стихийным бедствием.
- Сбор средств: сообщения описывают пожертвования денег и предметов пострадавшим от чрезвычайного происшествия.

Для того, чтобы классифицировать сообщения по типам и подтипам, в алгоритме используется наивный байесовский классификатор, который предварительно тренируется на размеченных данных. В классификаторе используется большое количество свойств, бинарных, скалярных и текстовых. В таблице 2 приведены результаты работы алгоритма на некоторых подтипах. Для тестирования использовались размеченные вручную сообщения.

Таблица 2. Результаты работы алгоритма information nuggets

Подтип	Точность	Полнота	F-мера
Предостережения и советы	0.618	0.598	0.605
Жертвы и разрушения	0.578	0.645	0.610
Сбор средств	0.546	0.632	0.585

2.4 Тематическая модель HDP

Как было показано в разделе 2.2, для того чтобы кластеризовать сообщения по темам, можно использовать тематические модели. Было обозначено, что примером та-

кой модели может быть модель LDA. Но в LDA необходимо указывать число тем как параметр, поэтому рассмотрим модель HDP (hierarchical Dirichlet process), которая автоматически находит подходящее число тем. В следующем разделе при описании решения будет указано, какие модификации были внесены в модель.

2.4.1 Иерархический процесс Дирихле

Пусть $(\Theta, \mathcal{B})$ — измеримое пространство, с вероятностной мерой G_0. Пусть α_0 — положительное действительное число. Определим процесс Дирихле $\mathrm{DP}(\alpha_0, G_0)$ как случайное распределение вероятностной меры G над $(\Theta, \mathcal{B})$, такое, что для любого конечного измеримого разбиения Θ $(A_1, A_2, \ldots, A_r)$, случайный вектор $(G(A_1), G(A_2), \ldots, G(A_r))$ будет распределен согласно конечномерному распределению Дирихле с параметрами $(\alpha_0 G_0(A_1), \alpha_0, G_0(A_2), \ldots, \alpha_0 G_0(A_r))$:

$$(G(A_1), G(A_2), \ldots, G(A_r)) \sim \mathrm{Dir}(\alpha_0 G_0(A_1), \alpha_0, G_0(A_2), \ldots, \alpha_0 G_0(A_r)). \tag{8}$$

Иерархический процесс Дирихле это распределение над множеством вероятностных мер над $(\Theta, \mathcal{B})$. Процесс определяет множество вероятностных мер G_j, по одному на каждую группу, и глобальную меру G_0. Глобальная мера G_0 распределена согласно процессу Дирихле с параметрами γ и базовым распределением H:

$$G_0 \mid \gamma, H \sim \mathrm{DP}(\gamma, H). \tag{9}$$

Групповые меры G_j условно независимы при заданном G_0 и имеют следующее распределение:

$$G_j \mid \alpha_0, G_0 \sim \mathrm{DP}(\alpha_0, G_0). \tag{10}$$

Гиперпараметрами HDP являются базовое распределение H и концентрационные параметры γ и α_0. Иерархический процесс Дирихле может быть использован как априорное распределение над параметрами групповых данных. Для каждого j пусть $\theta_{j1}, \theta_{j2}, \ldots$ — независимые распределенные по закону G_j случайные величины. Каждая величина θ_{ji} будет соответствовать наблюдаемому значению x_{ji}. Правдоподобие может быть записано в следующем виде:

$$\begin{aligned} &\theta_{ji} \mid G_j \sim G_j, \\ &x_{ji} \mid \theta_{ji} \sim F(\theta_{ji}). \end{aligned} \tag{11}$$

Процесс, описанные выше называется смешанной моделью для иерархического процесса Дирихле (hierarchical Dirichlet process mixture model) [8].

2.4.2 Chinese Restaurant Franchise

HDP — непараметрическая тематическая модель, процесс генерации данных по ней может быть описан в терминах процесса "chinese restaurant franchise"(CRF). В этом процессе существует сеть ресторана, в каждом из которых используется одинаковое меню. В ресторане находится неограниченное число столов, за каждым столом неограниченное число мест; каждый посетитель, заходя в ресторан, садится либо за уже занятый стол, либо за новый стол. На каждом непустом столе в ресторане заказано одно блюдо из меню, которое едят все посетители, сидящие за этим столом. Блюдо на стол заказывает первый человек, севший за этот стол.

В CRF группам соответствуют рестораны, а параметры θ_{ji} считаются посетителями. Переменные $\phi_1, \ldots, \phi_K$ составляют единое меню для сети ресторанов, они

распределены согласно базовому распределению H. Пусть ψ_{jt} — переменная, связанная со столом t в ресторане j и обозначающее блюдо, которое заказано за этим столом.

Каждому посетителю θ_{ji} соответствует стол ψ_{jt}, в то время как каждому столу ψ_{jt} соответствует одно блюдо ϕ_k. Для того, чтобы указать это соответствие, используются индексы:

- t_{ji} — индекс, связывающий посетителя θ_{ji} со столом ψ_{jt}, за которым сидит этот посетитель.
- k_{jt} — индекс, связывающий стол ψ_{jt} с блюдом ϕ_k, которое заказано для этого стола.

Для описания процесса, необходимо ввести следующие обозначения:

- n_{jtk} — количество посетителей в ресторане j за столом t, едящих блюдо k;
- $n_{jt\cdot}$ — количество посетителей в ресторане j за столом t;
- $n_{j\cdot k}$ — количество посетителей в ресторане j, едящих блюдо k;
- m_{jk} — количество столов в ресторане j, на которых заказано блюдо k;
- $m_{j\cdot}$ — количество столов в ресторане j;
- $m_{\cdot k}$ — количество столов, на которых заказано блюдо k;
- $m_{\cdot\cdot}$ — общее количество столов.

Запишем условное распределение θ_{ji}, где G_j исключены интегрированием:

$$\theta_{ji} \mid \theta_{j1}, \theta_{j2}, \ldots, \theta_{j,i-1}, \alpha_0, G_0 \sim \sum_{i=1}^{m_j} \frac{n_{jt\cdot}}{i-1+\alpha_0}\delta_{\psi_{jt}} + \frac{\alpha_0}{i-1+\alpha_0}G_0. \tag{12}$$

Аналогично, интегрируя по G_0, получим:

$$\psi_{jt} \mid \psi_{11}, \psi_{12}, \ldots, \psi_{21}, \ldots, \psi_{j,t-1}, \gamma, H \sim \sum_{k=1}^{K} \frac{m_{\cdot k}}{m_{\cdot\cdot}+\gamma}\delta_{\phi_k} + \frac{\gamma}{m_{\cdot\cdot}+\gamma}H. \tag{13}$$

Уравнение (12) описывает выбор стола посетителем, в то время как уравнение (13) определяет выбор блюда.

Если перевести используемые термины в область тематических моделей, посетители будут соответствовать словам, рестораны документам, столы внутренним темам документов, блюда внешним темам всего корпуса [8].

2.5 Нахождение экстремумов

Рассмотрим методы, которые могут быть использованы для того, чтобы найти экстремум функции. Как было замечено выше, события в сети Твиттер носят "взрывной" характер и соответствуют разкому повышению частоты сообщений. В связи с этим, нахождение экстремумов может быть использовано для поиска точек во времени, в которых могло произойти событие.

Понятие экстремума в этом подразделе отличается от привычного определения в математическом анализе. Назовем экстремумом те точки, в которых плотность сообщений позволяет утверждать что в этот момент произошло событие. В первую очередь, понятие экстремума для задачи выявления событий должно быть локальным. С другой стороны не все локальные максимумы можно корректно обозначить как события. Задача в общем виде получила развитие в контексте обработки сигналов [7]. Существует множество методов, каждый из которых будет больше подходить

к определенному типу данных. Алгоритм нахождения экстремумов может меняться вне зависимости от других составных частей исходного алгоритма. Несколько возможных подходов нахождения экстремумов для функции $f(x)$ с сеточной областью определения X перечислены ниже:

- Алгоритм помечает как экстремумы все точки, в которых функция достигает максимума в некотором окне, и при этом ее значение больше некоторого заранее определенного θ. Опционально θ может зависеть от среднего значения функции и других статистических характеристик.
- Для любых двух значений аргумента x и y, где $x < y$ определим функции $T(x, y)$ (travel) и $R(x, y)$ (rise):

$$\begin{aligned} T(x,y) &= \sum_{x \le k < y} |f(k+1) - f(k)|, \\ R(x,y) &= f(y) - f(x) + \epsilon, \end{aligned} \tag{14}$$

  где ϵ — некоторое небольшое значение. Тогда значения $T(x, y)/R(x, y) > 1$ соответствуют пику функции между точками x и y. Алгоритм может находить все экстремумы, где значение T/R больше некоторого порога.
- Подход заключается в том, чтобы выделить в функции стандартную пиковую подфункцию, например функцию Гаусса. Для этого применяются согласованные фильтры. Пусть $g(x)$ — функция Гаусса с некоторыми μ и σ:

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}. \tag{15}$$

  Тогда можно определить степень похожести исходной функции на функцию $g(x)$:

$$\mathrm{sim}(f, g) = \frac{(f, g)}{\|f\| \cdot \|g\|}. \tag{16}$$

  В косинусной метрике выражение примет следующий вид:

$$\mathrm{sim}(f, g) = \frac{\sum\limits_{x \in X} f(x) \cdot g(x)}{\sqrt{\sum\limits_{x \in X} f^2(x)} \cdot \sqrt{\sum\limits_{x \in X} g^2(x)}}. \tag{17}$$

  Аналогично предыдущим случаям, будем считать что функции “похожи” если значение $\mathrm{sim}(f, g)$ не меньше некоторого θ.
- Один из способов найти экстремумы — применить смешанную модель (mixture model) к входной функции. Компонентами модели могут быть нормальные распределения. Затем нужно найти параметры модели используя EM (expectation maximization) алгоритм или сэмплирование по Гиббсу. Этот метод будет корректно обрабатывать случай, когда экстремумы накладываются друг на друга или расположены очень близко.

3 Исследование и построение решения задачи

Эта работа ставит перед собой задачу разработать алгоритм нахождения событий в потоке сообщений пользователей сети Твиттер. В качестве данных были использованы сообщения с 4 июня 2013 года по 1 июля 2013 года, содержащие в себе хэштег #texas. Всего корпус включает порядка 240 тысяч сообщений и 1.5 миллиона слов. Приведем псевдокод алгоритма, а затем рассмотрим каждый шаг более подробно. Здесь и далее под частотной характеристикой сообщений подразумевается функция количества сообщений в час.

Исходные параметры: множество сообщений Ω, параметры алгоритма M, L, l_D, J, R, θ_{sim}, $\hat{\sigma}_0$, θ_{spam} параметры для HDP модели

Результат: для каждого события набор ключевых слов и частота сообщений по теме события

1 в частотной характеристике сообщений из Ω найти K экстремумов, пусть они соответствуют временам $t_1, t_2, \ldots, t_K$;
2 для каждого $t \in \{t_1, t_2, \ldots, t_K\}$ выполнять
3 извлечь M ключевых слов из сообщений в Ω в некотором радиусе R момента времени t;
4 сформировать фиктивные документы $D_0 = \left\{D_0^j\right\}_{j=1}^{J}$, каждый из которых имеет длину l_D и состоит из полученных на шаге 3 ключевых слов пропорционально их весу;
5 добавить D_0 к множеству Ω;
6 каждому сообщению из Ω назначить тему, с помощью модифицированной модели HDP (Hierarchical Dirichlet Process);
7 пусть документам D_0 была назначена тема k_0, удалить D_0 из Ω;
8 найти частотную характеристику сообщений в Ω с темой k_0;
9 если частотная характеристика определяет новое событие тогда
10 найти L ключевых слов темы k_0 из HDP;
11 добавить ключевые слова, частотную характеристику в результат;
12 конец условия
13 конец цикла

Алгоритм 1. Извлечение событий из сообщений сети Твиттер

3.1 Нахождение экстремумов

На шаге 1 основного алгоритма из частотной функции извлекаются экстремумы. Некоторые методы нахождения экстремумов были описаны в разделе 2.5. Их применение можно комбинировать со сглаживающими фильтрами если шум мешает выделить события. Для исходной задачи был выбран первый метод из списка, а именно нахождение максимума функции в некотором окне, при условии что значение функции выше пороговой величины. Данные показали что экстремумы сильно выделяются на фоне средней плотности сообщений и расположены на значительном расстоянии. Таким образом выбранный метод экспериментально обоснован для данных сети Твиттер.

3.2 Извлечение ключевых слов

Для того, чтобы использовать на шаге 6 модель HDP с частичным обучением, необходимо найти ключевые слова события, которые затем будут отслеживаться те-

матической моделью. Их извлечение происходит на шаге 3. В реализации алгоритма учитывая особенности сети Твиттер был выбран простой способ извлечь M ключевых слов из набора сообщений, а именно отсортировать слова по невозрастанию частоты в некотором окне и взять первых M слов.

Этот способ показал хорошие результаты потому что события в сети Твиттер носят "взрывной" характер и частота релевантных слов во время события в десятки и сотни раз превосходит нормальную частоту этих слов.

3.3 Модификация частичной обучаемости

HDP является моделью без учителя, она не требует размеченных данных для того чтобы назначить документам соответствующие темы. Для того, чтобы контролировать процесс определения тем, необходимо определенным образом изменить модель. В алгоритме 1 требуется добиться от HDP того, чтобы она выделила ключевые слова из некоторого множества $\mathcal{K}$ в отдельную тему k_0. Есть несколько способов достичь требуемого результата, один из них описан ниже.

Составим из ключевых слов в $\mathcal{K}$ документ длины M, в котором количество вхождений каждого ключевого слова пропорционально его весу. Перед тем как находить параметры HDP для корпуса документов, добавим J сгенерированных документов в корпус [10].

Для того, чтобы извлечь параметры из модели HDP в алгоритме применяется сэмплирование по Гиббсу. В этом методе изначальные документы каким-то образом распределяются по темам, а затем итеративно каждое слово удаляется из состояния и назначается на новое место, согласно уравнениям (12) и (13). Краткий псевдокод сэмплирования по Гиббсу для модели HDP приведен ниже:

Исходные параметры: множество документов Ω, параметры для HDP модели α_0, γ, I

```
назначить всем словам внутреннюю и внешнюю тему 0;
повторять
    для каждого документа d ∈ Ω выполнять
        для каждого слова w ∈ d выполнять
            удалить w из состояния модели;
            выбрать внутреннюю тему t согласно уравнению (12);
            если выбрана новая внутренняя тема t тогда
                выбрать внешнюю тему k согласно (13);
                назначить внутренней теме t внешнюю тему k;
            иначе
                извлечь назначенную ранее внешнюю тему k для внутренней
                темы t;
            конец условия
            назначить слову w внутреннюю тему t и внешнюю тему k;
        конец цикла
    конец цикла
до тех пор, пока не прошло I итераций;
```

Алгоритм 2. Сэмплирование по Гиббсу

Для того, чтобы внести в модель частичную обучаемость, необходимо по-разному обрабатывать оригинальные документы и дополнительные. Для оригинальных документов должен сохраниться первоначальный алгоритм, а словам в дополнительном

документе всегда будем назначать зафиксированную внешнюю тему k_0. Приведем псевдокод для модифицированной HDP:

Исходные параметры: множество документов Ω, параметры для HDP модели α_0, γ, k_0, I

```
назначить всем словам внутреннюю и внешнюю тему 0;
повторять
    для каждого документа d ∈ Ω выполнять
        для каждого слова w ∈ d выполнять
            если документ d дополнительный тогда
                назначить слову w внутреннюю тему 0 и внешнюю тему k_0;
            иначе
                удалить w из состояния модели;
                выбрать внутреннюю тему t согласно уравнению (12);
                если выбрана новая внутренняя тема t тогда
                    выбрать внешнюю тему k согласно (13);
                    назначить внутренней теме t внешнюю тему k;
                иначе
                    извлечь назначенную ранее внешнюю тему k для внутренней
                    темы t;
                конец условия
                назначить слову w внутреннюю тему t и внешнюю тему k;
            конец условия
        конец цикла
    конец цикла
до тех пор, пока не прошло I итераций;
```

Алгоритм 3. Сэмплирование по Гиббсу с частичным обучением

Изменяя параметры M и J можно определять насколько сильной будет привязка к конкретным ключевым словам для искомой темы. Экспериментально установлено что в качестве параметра k_0 можно взять значение 1.

3.4 Анализ сообщений темы

Возможны варианты, когда сообщения, найденные в экстремуме не являются новым событием. Это возможно в следующих случаях:

1. уже было найдено такое же событие,
2. график функции частоты не обладает свойствами, характерными событию.

Первый случай можно выявить, найдя нормализованное скалярное произведение двух функций частоты в косинусной метрике. Если значение будет близко к единице (больше параметра θ_{sim}), события можно считать тождественными. Чтобы различать второй случай, предлагается посчитать императивное стандартное отклонение, нормированное на максимальное значение частоты. Значение стандартного отклонения лучше всего считать на сглаженных данных. Если оно будет достаточно велико ($\hat{\sigma}^2 > \hat{\sigma}_0^2$), событие нельзя считать подходящим.

Полученный метод не будет работать в случае когда событие состоит из двух достаточно удаленных пиков. В этом случае можно использовать смешанную модель, как это было описанно выше в методах нахождения экстремума. При тестировании метода на реальных данных подобных событий не было найдено [11].

Дополнительно к частотным критериям необходимо ввести проверку, является ли событие спамом или информацией, интересной только узкому кругу пользователей. С большой долей точности можно судить о принадлежности события к спаму, если проверить отношение числа сообщений к числу авторов, которые отправляли эти сообщения [12]. Пусть событие содержит n сообщений от k авторов. Тогда если $k/n < \theta_{spam}$, можно утверждать, что событие является спамом. Параметр θ_{spam} можно выяснить, посчитав отношение k/n на тестовых данных. В реализации алгоритма выбрано значение $\theta_{spam} = 0.05$.

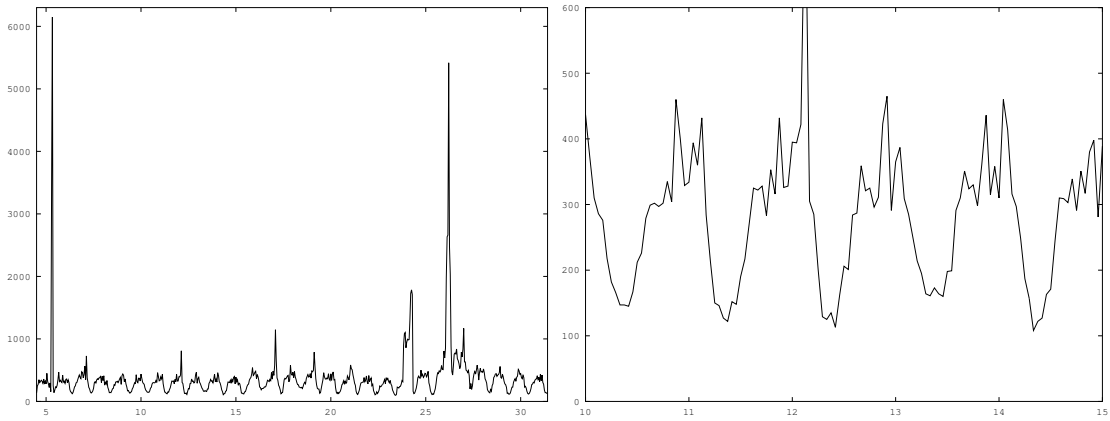
4 Реализация и тестирование решения

4.1 Описание данных

В качестве данных для задачи были использованы сообщения пользователей сети Твиттер с 4 июня по 1 июля 2013 года, содержащие хэштег #texas. Всего в корпусе находится около 240 тысяч сообщений. Цель алгоритма — найти события, показать их на графике частоты сообщений и указать ключевые слова к событиям.

Изначально данные были предобработаны: удалены знаки препинания, слова приведены к одному регистру, применен стемминг.

На рисунке 2 изображена плотность сообщений во входных данных. На графике есть несколько пиков, которые обозначают события.



а) б)

Рисунок 2. Зависимость количества сообщений в час от времени. а) Полный временной отрезок. б) Зависимость частоты от времени суток

4.2 Извлечение событий

На рисунке 3 можно видеть экстремумы, отмеченные алгоритмом 1 на шаге 1. Путем выбора метода и параметров, можно помечать больше или меньше максимумов.

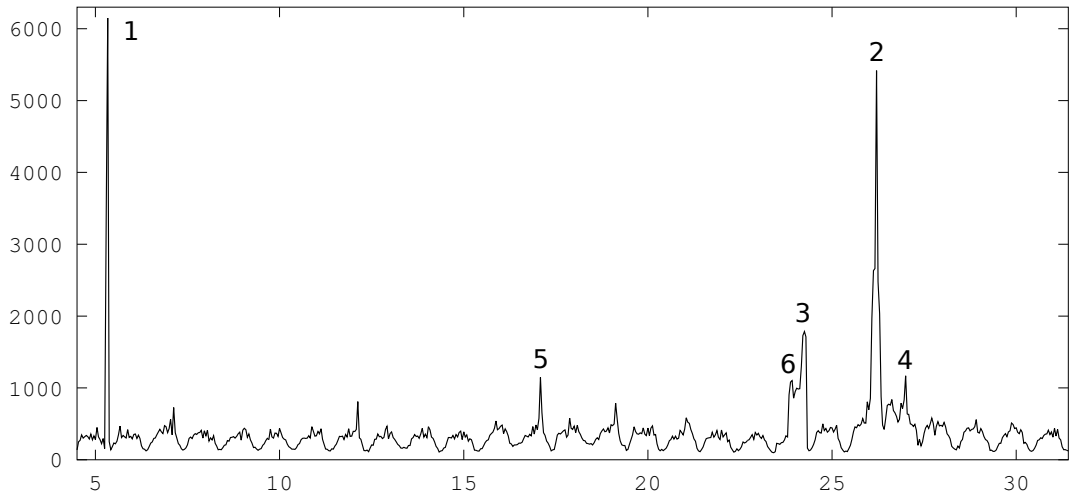


Рисунок 3. График частоты сообщений с отмеченными экстремумами

Результатом работы тематической модели на шаге 6 являются размеченные сообщения. На графиках показаны частоты сообщений, соответствующие рассматриваемым событиям.

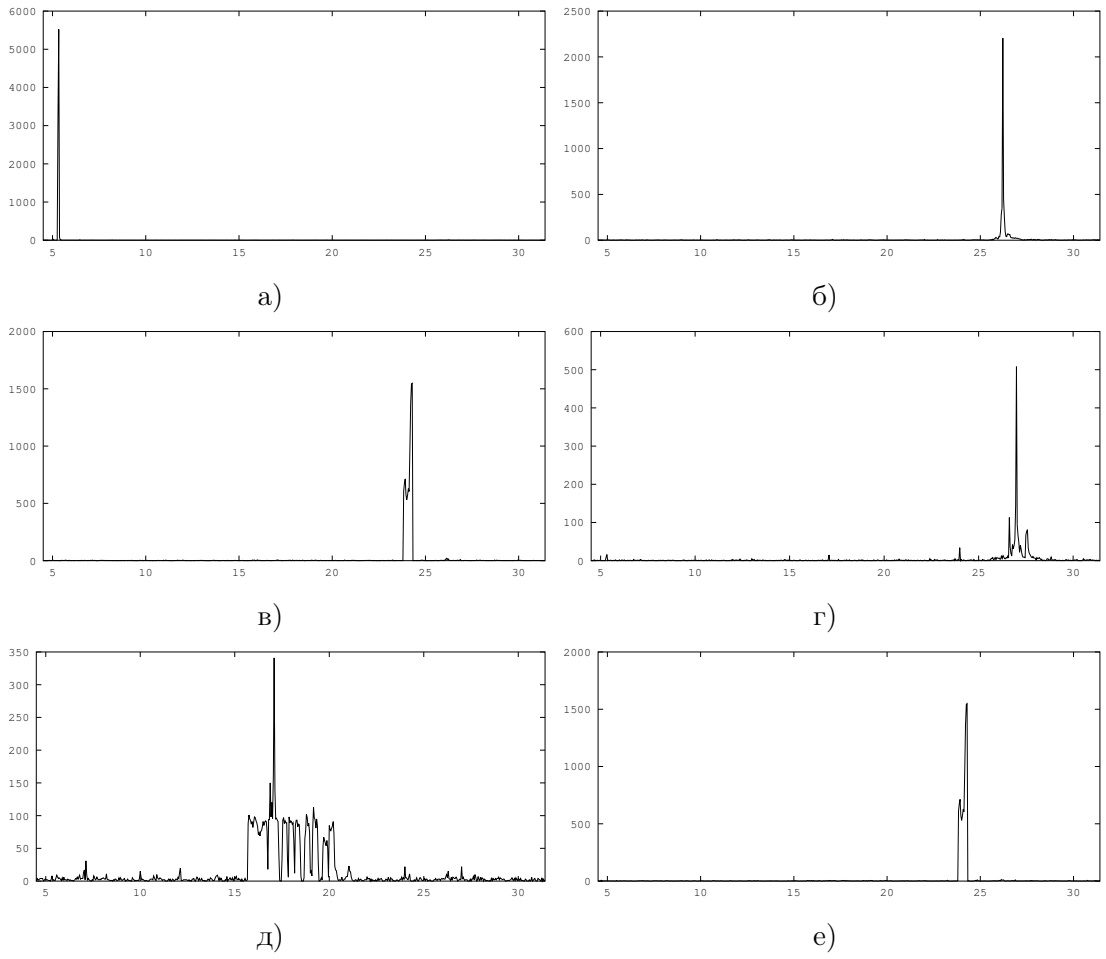


Рисунок 4. Частоты сообщений для “событий” 1 — 6. а) Событие 1. б) Событие 2. в) Событие 3. г) Событие 4. д) Событие 5. е) Событие 6.

Следующие таблицы показывают параметры частотных характеристик сообщений, полученных на шаге 8 основного алгоритма. На основании этих данных те или иные события войдут в финальный результат или будут отброшены.

Таблица 3. Значения нормализованного среднеквадратичного отклонения. Помечены события, для которых $\hat{\sigma}^2 > \hat{\sigma}_0^2$

	$\hat{\sigma}^2$
1	0.00004
2	0.00188
3	0.00831
4	0.34469
5	**3.81528**
6	0.00831

Таблица 4. Значения нормализованного скалярного произведения функций частот в косинусной метрике. Помечены значения, позволяющие считать события тождественными

1	1					
2	0.004	1				
3	0.001	0.009	1			
4	0.049	0.052	0.018	1		
5	0.011	0.022	0.027	0.050	1	
6	0.001	0.007	**1.0**	0.019	0.026	1

Таблица 5. Отношение количества уникальных авторов к общему числу соощений

	k/n
1	0.992
2	0.967
3	0.983
4	0.855
5	0.178
6	0.983

Из таблиц 3 и 4 можно видеть, что алгоритм выдаст в конечном счете только события 1, 2, 3, 4. Событие 5 будет отброшено как имеющее большое нормализованное среднеквадратичное отклонение. Событие 6 не войдет в финальный результат потому, что значение похожести функций частоты для событий 3 и 6 больше допустимой нормы:

$$\mathrm{sim}^{\cos}(f_3, f_6) = \frac{(f_3, f_6)}{\|f_3\| \cdot \|f_6\|} > \theta_{\mathrm{sim}}. \tag{18}$$

Приведем ниже таблицу 6 с описаниями событий 1, 2, 3, 4:

Таблица 6. Таблица с описаниями событий

№	Ключевые слова	Вызвавшее событие
1	year, read, prison, dragon, amazon, written, bestseller	Написанная в техасской тюрьме книга The Sword and the Dragon является бестселлером на Amazon уже более трех лет.
2	sb5, standwithwendi, abort, senat	Обсуждение сенатом закона по запрету абортов, который носит название Senate Bill 5. Wendi Davis — политик, которая боролась с принятием этого закона.
3	houston, california, england, artist, rt, watch, defjam	Хип хоп исполнитель из Хьюстон Devin the Dude объявил, что его восьмой студийный альбом будет называться One for the Road и будет выпущен в продажу в сентябре 2013 года. Альбом будет выпущен лейблом Def Jam Recordings.
4	execut, 500th, mccarthi, kimberli, 1976	В штате Техас приведен в исполнение 500-й по счету смертный приговор. Смертная казнь имеет место с 1976 года.

Ниже указаны значения параметров алгоритма 1, при которых были получены результаты этого раздела:

Таблица 7. Использованные значения параметров

M	20
L	20
l_D	50
J	10^4
R	30 минут
I	300
α_0	1.0
γ	2.0
k_0	1
θ_{sim}	0.95
$\hat{\sigma}_0^2$	3
θ_{spam}	0.05

Помимо алгоритма 1 на данных был запущен оптимизированный алгоритм New event detection, описанный в разделе 2.1. Оптимизация состоит в том, что при поиске наиболее близкого документа для заданного документа d, проверка проходила не по всем предыдущим документам, а только по 20000 последних. В результате было выяснено что NED помимо правильно помеченных сообщений также отмечает множество документов, которые не являются событием.

Это происходит потому, что сеть Твиттер содержит большое количество шума, который нередко состоит из слов, нигде больше не встречающихся. Примером

таких сообщений может быть “But #Texas ... #Why???Lol #wheredeydodatat Oh. #Dallas#Shaq #SodaShaq ??”. NED будет помечать эти сообщения новыми событиями, хотя они таковыми не являются. Алгоритм 1 лишен этого недостатка потому что на первом шаге рассматривает максимумы частотной характеристики, значение частоты сообщений в которых больше нормального уровня шума. Затем в случае если этот максимум составлен шумом или спамом, это будет выявлено на шаге 8.

4.3 Оценка качества разработанного алгоритма

Для того, чтобы иметь возможность сравнивать алгоритм с аналогами, необходимо предложить критерий качества полученного результата. В данной работе предлагается использовать точность, полноту и F-меру алгоритма как метод оценки качества алгоритмов.

Пусть алгоритм на выходе выдал информацию об n найденных событиях, из которых k событий определены верно. Тогда точностью алгоритма будет называться величина $p = k/n \in [0,\, 1]$.

Для корректного сравнения алгоритмов помимо точности, необходимо использовать полноту. Пусть в корпусе Ω содержится информация об m событиях, из них l событий были успешно отмечены алгоритмом. Тогда полнотой алгоритма называется отношение $r = l/m \in [0,\, 1]$. F-мерой алгоритма называется среднее гармоническое точности и полноты (считается, что $p \cdot q > 0$):

$$F = \left(\frac{1}{p} + \frac{1}{r}\right)^{-1} = \left(\frac{n}{k} + \frac{m}{l}\right)^{-1}. \tag{19}$$

Очевидно, что чем выше значения p, q и F показывает алгоритм на определенных данных, тем лучше он работает на этих данных.

Заметим, что для определения точности необходимо вручную проверить n событий, выданных алгоритмом, и определить насколько они достоверные, а для определения полноты просмотреть все $|\Omega|$ сообщений и выделить из них события. В связи с этим, для того, чтобы упростить определение полноты алгоритма будем считать, что событие может содержаться в корпусе только в той точке, частота сообщений в которой является максимумом в некотором окне и величина частоты больше значения θ_{freq}. Это предположение означает что нас интересуют только те события, которые активно обсуждались пользователями сети Твиттер. При тестировании решения θ_{freq} было взято средним значением частоты сообщений.

Ниже указана таблица со значениями точности, полноты и F-меры для разработанного алгоритма и для NED. По таблице можно видеть что алгоритм 1 работает на данных сети Твиттер существенно лучше алгоритма NED.

Таблица 8. Оценка предложенного алгоритма и алгоритма New event detection на имеющихся данных

Алгоритм	Точность	Полнота	F-мера
Алгоритм 1	0.8	1.0	0.44
NED	0.01	0.8	0.01

5 Описание практической части

Для реализации программы, описанной алгоритмом 1, был выбран язык Java. Выбор мотивирован тем, что в качестве свободной реализации для модели HDP использовалась реализация на языке Java, написанная Bleier et al [9]. Java относится к объектно-ориентированным языкам высокого уровня, что оказалось очень удобным для реализации алгоритма.

Приведем ниже диаграмму классов для модуля, в котором находится логика алгоритма и краткое описание классов программы.

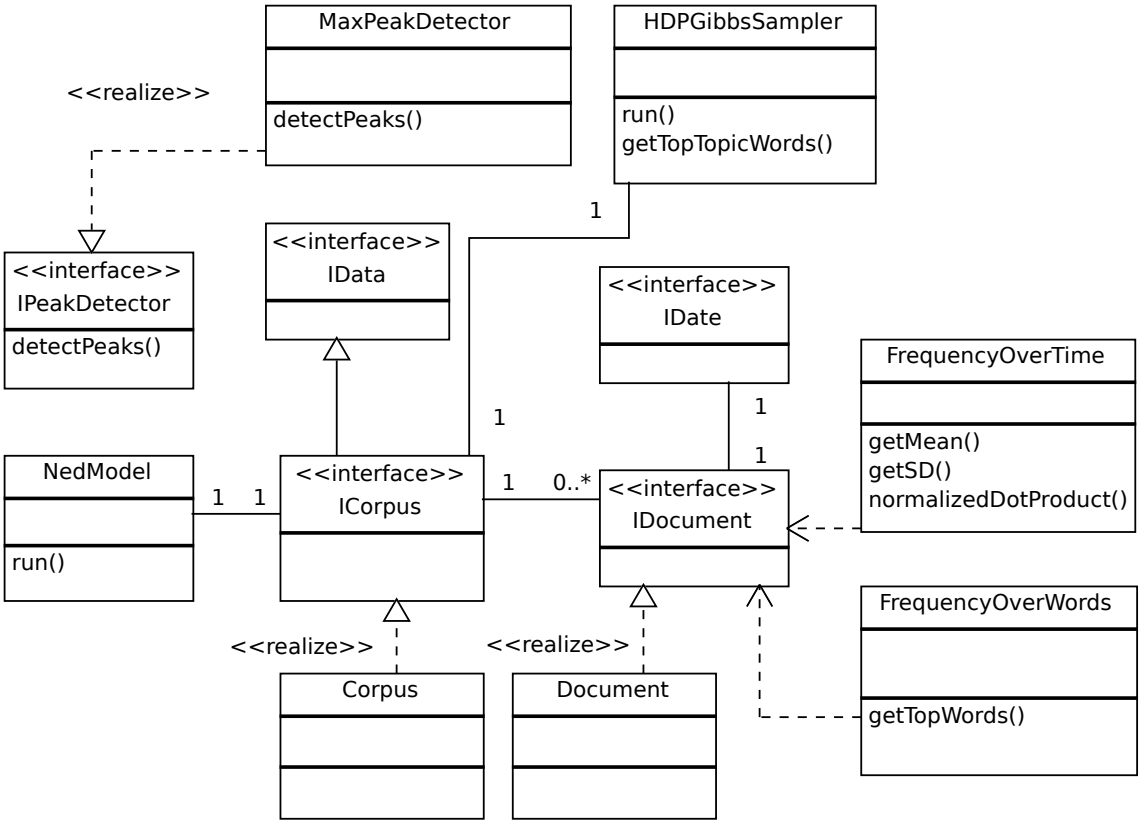


Рисунок 5. Диаграмма классов для пакета hdp

- CollectionUtils: утилитарный класс для работы с коллекциями.
- Corpus: класс, содержащий в себе документы для задачи.
- Date: класс, работающий с датой. Используется при работе с частотой сообщений.
- DateUtils: утилитарный класс для работы с датой.
- Document: документ, характеризуется словами, датой, автором и темой.
- FrequencyOverTime: класс, работающий с частотной характеристикой сообщений во времени.
- FrequencyOverWords: класс, который извлекает ключевые слова из документов.
- HDPGibbsSampler: модель HDP с модификацией частичного обучения. Размечает документы соответствующими темами.
- Main: здесь находится точка входа программы, в методе main реализуется разработанный алгоритм.

– MaxPeakDetector: класс, находящий экстремумы в частотной функции сообщений.
– MiscUtils: утилитарный класс.
– NedModel: модифицированная модель NED, используется для сравнения работы с разработанным алгоритмом.
– Pair: утилитарный класс для работы с парами объектов.
– ProbUtils: утилитарный класс для работы с вероятностями.
– WordProp: утилитарный класс, используется в деталях реализации.

Полный текст программы, реализующей разработанный алгоритм можно найти в приложении.

Заключение

В ходе выполения дипломной работы было достигнуто:

– изучены существующие методы извлечения описаний событий из коротких пользовательских сообщений,
– изучена возможность применения тематических моделей для этой задачи,
– разработан и проанализирован алгоритм извлечения описаний событий из социальной сети Твиттер на основе иерархического процесса Дирихле,
– алгоритм был реализован, параметры подобраны экспериментальным путем,
– тестирование показало, что алгоритм успешно распознает события, имеющиеся во входных данных.

Дальнейшее улучшение алгоритма можно проводить в следующих направлениях:

– исследовать работу метода на данных с большим временным отрезком, добавить возможность обработки событий с несколькими удаленными максимумами в функции частоты,
– добавить возможность извлечения подсобытий,
– использовать информацию об авторах, геотеги сообщений и другие дополнительные данные для более точного извлечения описаний событий.

Список литературы

1. Imran, Elbassuoni, Castillo, Diaz and Meier. Extracting Information Nuggets from Disaster-Related Messages in Social Media // 2013.

2. Xun Wang, Feida Zhu, Jing Jiang, Sujian Li. Real Time Event Detection in Twitter // 2011.

3. Thorsten Brants, Francine Chen, Ayman Farahat. A System for New Event Detection // 2003.

4. Konstantinos N. Vavliakis, Fani A. Tzima, and Pericles A. Mitkas. Event Detection via LDA for the MediaEval2012 SED Task // 2012.

5. David M. Blei, Andrew Y. Ng, Michael I. Jordan. Latent Dirichlet Allocation // 2003.

6. Tom Griffiths. Gibbs sampling in the generative model of Latent Dirichlet Allocation.

7. Girish Keshav Palshikar. Simple Algorithms for Peak Detection in Time-Series.

8. Yee Whye Teh, Michael I. Jordan, Matthew J. Beal, David M. Blei. Hierarchical Dirichlet Processes // 2005.

9. Yee Whye Teh. Hierarchical Bayesian Nonparametric Models with Applications // 2009.

10. Ramnath Balasubramanyan, William W. Cohen, Matthew Hurst. Modeling corpora of timestamped documents using semisupervised nonparametric topic models.

11. David Blei. COS 424: Interacting with Data // 2008.

12. Sasa Petrovic, Miles Osborne, Victor Lavrenko. Streaming First Story Detection with application to Twitter // 2010.

Приложение

В приложении приведен исходный код разработанного алгоритма.

```
package net.msusevastopol.math.ypys.utils;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;

public class CollectionUtils
{
   // Prevent instantiating.
   private CollectionUtils()
   {
   }

   public static <T> List<T> newList()
   {
      return new ArrayList<T>();
   }

   public static <K, V> Map<K, V> newMap()
   {
      return new HashMap<K, V>();
   }

   public static <K> Set<K> newSet()
   {
      return new HashSet<K>();
   }

   public static <K, V> V getDefault(Map<K, V> map, K key, V defaultValue)
   {
      if (map.containsKey(key))
         return map.get(key);
      return defaultValue;
   }

   public static <K, V> void initMap(Map<K, V> map, K key, V value)
   {
      if (map.containsKey(key))
         return;
      map.put(key, value);
   }
}
```

```
package net.msusevastopol.math.ypys.hdp;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.util.Iterator;
import java.util.List;
import java.util.Set;

import net.msusevastopol.math.ypys.utils.CollectionUtils;
import net.msusevastopol.math.ypys.utils.MiscUtils;

public class Corpus implements ICorpus
{

   /*
    * [THEME) ]WORD1 WORD2 ... WORDN | DAY HOUR | AUTHOR
    */

   private List<IDocument> documents = CollectionUtils.newList();

   public Corpus()
   {
   }

   public Corpus(String filename)
   {
      try
      {
         load(filename);
      }
      catch (FileNotFoundException e)
      {
         e.printStackTrace();
      }
      catch (IOException e)
      {
         e.printStackTrace();
      }
   }

   @Override
   public void save(String filename) throws IOException, FileNotFoundException
   {
      PrintWriter out = new PrintWriter(filename);

      for (IDocument document : documents)
```

```
   {
      if (document.getTopic() == null)
         out.format("%s | %d %d | %s\n", MiscUtils.implode(" ",
               document.getWords()), document.getDate().getDay(),
               document.getDate().getHour(), document.getAuthor());
      else
         out.format("%d) %s | %d %d | %s\n", document.getTopic(),
               MiscUtils.implode(" ", document.getWords()), document
                     .getDate().getDay(), document.getDate()
                     .getHour(), document.getAuthor());
   }

   out.flush();
   out.close();
}

@Override
public void load(String filename) throws IOException, FileNotFoundException
{
   BufferedReader in = new BufferedReader(new InputStreamReader(
         new FileInputStream(filename)));

   while (in.ready())
   {
      String line = in.readLine();
      String[] words = ParsingUtils.extractWords(line);
      Integer topic = ParsingUtils.extractTheme(line);
      IDate date = ParsingUtils.extractDate(line);
      String author = ParsingUtils.extractAuthor(line);

      documents.add(new Document(words, topic, date, author));
   }

   in.close();
}

@Override
public List<IDocument> getDocuments()
{
   return documents;
}

private static class ParsingUtils
{
   // Prevent instantiating.
   private ParsingUtils()
   {
   }

   public static IDate extractDate(String line)
```

```
   {
      return new Date(line.split(" \\| ")[1]);
   }

   public static String extractAuthor(String line)
   {
      return line.split(" \\| ")[2];
   }

   public static String[] extractWords(String line)
   {
      if (line.indexOf(")") >= 0)
         return line.split("\\)")[1].split(" \\| ")[0].trim().split(" ");
      return line.split(" \\| ")[0].trim().split(" ");
   }

   public static Integer extractTheme(String line)
   {
      if (line.indexOf(")") >= 0)
         return Integer.parseInt(line.split("\\)")[0]);
      return null;
   }
}

@Override
public void addAdditionalDocument(IDocument document)
{
   documents.add(0, document);
}

@Override
public void clearAdditionalDocuments()
{
   for (Iterator<IDocument> it = documents.iterator(); it.hasNext();)
   {
      IDocument document = it.next();
      if (document.isAdditional())
         it.remove();
      else
         break;
   }
}

@Override
public double printUniqueAuthorsProportionForTopic(Integer topic)
{
   int documentsCount = 0;
   Set<String> authors = CollectionUtils.newSet();
   for (IDocument document : this.getDocuments())
   {
```

```
         if (topic == null || topic.equals(document.getTopic()))
         {
            authors.add(document.getAuthor());
            documentsCount++;
         }
      }

      return ((double) authors.size()) / documentsCount;
   }
}
```

```
package net.msusevastopol.math.ypys.hdp;

public class Date implements IDate
{
   private int day;
   private int hour;

   public Date(String line)
   {
      day = Integer.parseInt(line.split(" ")[0]);
      hour = Integer.parseInt(line.split(" ")[1]);
   }

   public Date(int day, int hour)
   {
      this.day = day;
      this.hour = hour;
   }

   public int getDay()
   {
      return day;
   }

   public int getHour()
   {
      return hour;
   }

   @Override
   public String toString()
   {
      return "Date [day=" + day + ", hour=" + hour + "]";
   }

   @Override
   public int hashCode()
   {
```

```
      final int prime = 31;
      int result = 1;
      result = prime * result + day;
      result = prime * result + hour;
      return result;
   }

   @Override
   public boolean equals(Object obj)
   {
      if (this == obj)
         return true;
      if (obj == null)
         return false;
      if (getClass() != obj.getClass())
         return false;
      Date other = (Date) obj;
      if (day != other.day)
         return false;
      if (hour != other.hour)
         return false;
      return true;
   }

   @Override
   public String toShortString()
   {
      return "(" + day + "-" + hour + ")";
   }

}
```

```
package net.msusevastopol.math.ypys.utils;

import net.msusevastopol.math.ypys.hdp.Date;
import net.msusevastopol.math.ypys.hdp.IDate;

public class DateUtils
{
   // Prevent instantiating.
   private DateUtils()
   {
   }

   private static final int OFFSET = 12 + 4 * 24;

   public static int toCode(IDate date)
   {
      return date.getDay() * 24 + date.getHour() - OFFSET;
```

```
   }

   public static IDate fromCode(int code)
   {
      code += OFFSET;
      return new Date(code / 24, code % 24);
   }

   public static int diffHours(IDate a, IDate b)
   {
      return Math.abs(toCode(a) - toCode(b));
   }
}
```

```
package net.msusevastopol.math.ypys.hdp;

import java.util.Arrays;
import java.util.List;

import net.msusevastopol.math.ypys.utils.CollectionUtils;
import net.msusevastopol.math.ypys.utils.MiscUtils;

public class Document implements IDocument
{
   private String[] words;
   private Integer topic;
   private IDate date;
   private String author;
   private static String ADDITIONAL = "Additional";

   public Document(String[] words, Integer topic, IDate date, String author)
   {
      this.words = words;
      this.setTopic(topic);
      this.date = date;
      this.author = author;
   }

   public Document(String[] words, IDate date)
   {
      this(words, null, date, "Default");
   }

   public String[] getWords()
   {
      return words;
   }

   public Integer getTopic()
```

```
   {
      return topic;
   }

   public void setTopic(Integer topic)
   {
      this.topic = topic;
   }

   public IDate getDate()
   {
      return date;
   }

   @Override
   public String toString()
   {
      return "Document [words=" + Arrays.toString(words) + ", topic="
            + getTopic() + ", date=" + date + "]";
   }

   public static IDocument getFromProp(WordProp[] prop, int length)
   {
      MiscUtils.normalize(prop);
      List<String> words = CollectionUtils.newList();
      for (WordProp wp : prop)
      {
         int amount = (int) Math.round(wp.prop * length);
         for (int i = 0; i < amount; i++)
            words.add(wp.word);
      }

      return new Document(words.toArray(new String[0]), null, IDate.ZERO,
            ADDITIONAL);
   }

   @Override
   public boolean isAdditional()
   {
      return ADDITIONAL.equals(author);
   }

   public String getAuthor()
   {
      return author;
   }
}
```

```
package net.msusevastopol.math.ypys.hdp;
```

```
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.Map;

import net.msusevastopol.math.ypys.utils.CollectionUtils;
import net.msusevastopol.math.ypys.utils.DateUtils;

public class FrequencyOverTime
{
   private int[] frequency;
   private Map<IDate, Integer> mapping;
   private static final int SOME_MINIMAL_VALUE_FOR_SMOOTHING = 30;

   public FrequencyOverTime(ICorpus corpus)
   {
      this(corpus, null);
   }

   public FrequencyOverTime(ICorpus corpus, Integer topic)
   {
      IDate last = corpus.getDocuments()
            .get(corpus.getDocuments().size() - 1).getDate();
      frequency = new int[DateUtils.toCode(last) + 1];

      for (IDocument document : corpus.getDocuments())
      {
         if (!document.isAdditional()
               && (null == topic || topic.equals(document.getTopic())))
            frequency[DateUtils.toCode(document.getDate())]++;
      }

      mapping = CollectionUtils.newMap();
      for (int i = 0; i < frequency.length; i++)
         mapping.put(DateUtils.fromCode(i), frequency[i]);
   }

   public void save(String filename) throws FileNotFoundException
   {
      PrintWriter out = new PrintWriter(filename);

      for (int value : frequency)
         out.println(value);

      out.flush();
      out.close();
   }
```

```
public double getMean()
{
   BigDecimal result = BigDecimal.ZERO;
   long sum = 0;
   for (int i = 0; i < frequency.length; i++)
   {
      if (frequency[i] <= SOME_MINIMAL_VALUE_FOR_SMOOTHING)
         continue;

      result = result.add(BigDecimal.valueOf((long) frequency[i] * i));
      sum += frequency[i];
   }
   return result.divide(BigDecimal.valueOf(sum), 50, RoundingMode.HALF_UP)
         .doubleValue();
}

public double getSD2()
{
   double mean = getMean();
   BigDecimal result = BigDecimal.ZERO;
   long sum = 0;
   long max = 0;
   for (int i = 1; i < frequency.length - 1; i++)
   {
      int value = frequency[i];

      if (frequency[i] <= SOME_MINIMAL_VALUE_FOR_SMOOTHING)
         continue;

      BigDecimal tmp = BigDecimal.valueOf(i - mean);
      result = result.add(BigDecimal.valueOf(value).multiply(tmp)
            .multiply(tmp));
      sum += value;
      max = Math.max(value, max);
   }
   return result.divide(BigDecimal.valueOf(sum), 50, RoundingMode.HALF_UP)
         .divide(new BigDecimal(max), 50, RoundingMode.HALF_UP)
         .doubleValue();
}

public Map<IDate, Integer> getMapping()
{
   return mapping;
}

public double normalizedDotProduct(FrequencyOverTime b)
{
   return this.dotProduct(b) / this.norm() / b.norm();
}
```

```
   private double norm()
   {
      return Math.sqrt(this.dotProduct(this));
   }

   private long dotProduct(FrequencyOverTime b)
   {
      if (this.frequency.length != b.frequency.length)
         throw new IllegalArgumentException();

      long ret = 0;
      for (int i = 0; i < frequency.length; i++)
      {
         ret += frequency[i] * b.frequency[i];
      }
      return ret;
   }
}
```

```
package net.msusevastopol.math.ypys.hdp;

import java.util.Arrays;
import java.util.Comparator;
import java.util.List;
import java.util.Map;

import net.msusevastopol.math.ypys.utils.CollectionUtils;
import net.msusevastopol.math.ypys.utils.DateUtils;

public class FrequencyOverWords
{
   private Map<String, Double> prop;

   public FrequencyOverWords(Map<String, Double> prop)
   {
      this.prop = prop;
   }

   public FrequencyOverWords()
   {
      prop = CollectionUtils.newMap();
   }

   public void add(String word)
   {
      if (prop.containsKey(word))
         prop.put(word, prop.get(word) + 1);
      else
         prop.put(word, 1.0);
```

```
}

private void normalize()
{
   double sum = 0.0;
   for (Double v : prop.values())
      sum += v;

   for (String word : prop.keySet())
      prop.put(word, prop.get(word) / sum);
}

/**
 * Assumed that P contains this.
 *
 */
public FrequencyOverWords diff(FrequencyOverWords p)
{
   return FrequencyOverWords.diff(this, p);
}

private static FrequencyOverWords diff(FrequencyOverWords a,
      FrequencyOverWords b)
{
   a.normalize();
   b.normalize();

   FrequencyOverWords c = new FrequencyOverWords();
   for (String word : a.prop.keySet())
      c.prop.put(word, -a.prop.get(word) * Math.log(b.prop.get(word)));

   c.normalize();
   return c;
}

public String[] getTopWords(int top)
{
   WordProp[] result = getTopWordsExtended(top);
   String[] strings = new String[result.length];

   for (int i = 0; i < result.length; i++)
      strings[i] = result[i].word;

   return strings;
}

public WordProp[] getTopWordsExtended(int top)
{
   WordProp[] result = new WordProp[prop.size()];
   int index = 0;
```

```
      for (String word : prop.keySet())
         result[index++] = new WordProp(word, prop.get(word));

      Arrays.sort(result, new Comparator<WordProp>()
      {

         public int compare(WordProp a, WordProp b)
         {
            return Double.valueOf(b.prop).compareTo(a.prop);
         }
      });

      return Arrays.copyOf(result, Math.min(result.length, top));
   }

   public static FrequencyOverWords getFromCorpus(ICorpus corpus,
         Integer topic, IDate date, int radius)
   {
      FrequencyOverWords result = new FrequencyOverWords();
      List<IDocument> documents = corpus.getDocuments();

      for (IDocument document : documents)
         if ((null == topic || topic.equals(document.getTopic()))
               && (null == date || inRange(date, document.getDate(),
                     radius)))
            for (String word : document.getWords())
               result.add(word);

      return result;
   }

   private static boolean inRange(IDate date, IDate date2, int radius)
   {
      return DateUtils.diffHours(date, date2) <= radius;
   }

   public static FrequencyOverWords getFromCorpusWithWeirdTfIdf(ICorpus corpus,
         Integer topic, IDate date, int radius)
   {
      FrequencyOverWords dateSpecific = getFromCorpus(corpus, topic, date,
            radius);
      FrequencyOverWords all = getFromCorpus(corpus, topic, null, 0);

      return dateSpecific.diff(all);
   }
}
```

```
/*
 * Copyright 2011 Arnim Bleier, Andreas Niekler and Patrick Jaehnichen
```

```
 * Licensed under the GNU Lesser General Public License.
 * http://www.gnu.org/licenses/lgpl.html
 */

package net.msusevastopol.math.ypys.hdp;

import java.io.IOException;
import java.io.PrintWriter;
import java.io.Serializable;
import java.util.Arrays;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
import java.util.Map;
import java.util.Random;

import net.msusevastopol.math.ypys.utils.CollectionUtils;
import net.msusevastopol.math.ypys.utils.ProbUtils;

/**
 * Hierarchical Dirichlet Processes Chinese Restaurant Franchise Sampler
 *
 * For more information on the algorithm see: Hierarchical Bayesian
 * Nonparametric Models with Applications. Y.W. Teh and M.I. Jordan. Bayesian
 * Nonparametrics, 2010. Cambridge University Press.
 * http://www.gatsby.ucl.ac.uk/~ywteh/research/npbayes/TehJor2010a.pdf
 *
 * For other known implementations see README.txt
 *
 * @author <a href="mailto:arnim.bleier+hdp@gmail.com">Arnim Bleier</a>;
 *         Modified by Yukhym Pyshnohraiev.
 *
 */
public class HDPGibbsSampler implements Serializable
{
   private boolean isDocumentAdditional(int d)
   {
      return corpus.getDocuments().get(d).isAdditional();
   }

   private PrintWriter log;

   private static final long serialVersionUID = 1L;
   private double beta = 0.5; // default only
   private double gamma = 2.0;
   private double alpha = 1.0;

   private Random random = new Random();
   private double[] p;
   private double[] f;
```

```
private DOCState[] docStates;
private int[] numberOfTablesByTopic;
private int[] wordCountByTopic;
private int[][] wordCountByTopicAndTerm;

private int sizeOfVocabulary;
private int numberOfTopics;
private int totalNumberOfTables;

private Map<Integer, String> codesToWords;

private ICorpus corpus;

public HDPGibbsSampler(ICorpus corpus, PrintWriter log)
{
   addInstances(corpus, log);
}

public void run(int maxIter) throws IOException
{
   run(0, maxIter, log);
   setTopicsToCorpus();
}

private void addInstances(ICorpus corpus, PrintWriter log)
{
   this.log = log;
   this.corpus = corpus;
   Map<String, Integer> codesAssignment = CollectionUtils.newMap();
   codesToWords = CollectionUtils.newMap();

   int[][] documents = new int[corpus.getDocuments().size()][];
   int code = 0;
   for (int d = 0; d < corpus.getDocuments().size(); d++)
   {
      documents[d] = new int[corpus.getDocuments().get(d).getWords().length];
      for (int w = 0; w < corpus.getDocuments().get(d).getWords().length; w++)
      {
         String word = corpus.getDocuments().get(d).getWords()[w];
         if (codesAssignment.containsKey(word))
         {
            documents[d][w] = codesAssignment.get(word);
         }
         else
         {
            codesAssignment.put(word, code);
            codesToWords.put(code, word);
            documents[d][w] = code++;
         }
```

```
      }
   }

   addInstances(documents, code);
}

private void setTopicsToCorpus()
{
   for (int d = 0; d < docStates.length; d++)
      corpus.getDocuments().get(d).setTopic(docStates[d].resolveTopic());
}

/**
 * Initially assign the words to tables and topics
 *
 * @param corpus
 *           {@link CLDACorpus} on which to fit the model
 */
private void addInstances(int[][] documentsInput, int V)
{
   sizeOfVocabulary = V;
   numberOfTopics = 1;
   docStates = new DOCState[documentsInput.length];
   for (int d = 0; d < documentsInput.length; d++)
   {
      docStates[d] = new DOCState(documentsInput[d], d);
   }
   int k, i, j;
   DOCState docState;
   p = new double[20];
   f = new double[20];
   numberOfTablesByTopic = new int[getNumberOfTopics() + 1];
   wordCountByTopic = new int[getNumberOfTopics() + 1];
   wordCountByTopicAndTerm = new int[getNumberOfTopics() + 1][];
   for (k = 0; k <= getNumberOfTopics(); k++)
      // var initialization done
      wordCountByTopicAndTerm[k] = new int[sizeOfVocabulary];
   for (k = 0; k < getNumberOfTopics(); k++)
   {
      docState = docStates[k];
      for (i = 0; i < docState.documentLength; i++)
         addWord(docState.docID, i, 0, k);
   } // all topics have now one document
   for (j = getNumberOfTopics(); j < docStates.length; j++)
   {
      docState = docStates[j];
      k = random.nextInt(getNumberOfTopics());
      for (i = 0; i < docState.documentLength; i++)
         addWord(docState.docID, i, 0, k);
   } // the words in the remaining documents are now assigned too
```

```
}

/**
 * Step one step ahead
 *
 */
private void nextGibbsSweep()
{
   for (int d = 0; d < docStates.length; d++)
   {
      for (int i = 0; i < docStates[d].documentLength; i++)
      {
         if (isDocumentAdditional(d))
            sweepForAdditionalDocument(d, i);
         else
            sweepForNormalDocument(d, i);
         // Table
      }
   }
   defragment();
}

private void sweepForNormalDocument(int d, int i)
{
   removeWord(d, i); // remove the word i from the state
   int table = sampleTable(d, i);
   if (table == docStates[d].numberOfTables) // new Table
   {
      int topic = sampleTopic();
      addWord(d, i, table, topic); // sampling its
   }
   else
      addWord(d, i, table, docStates[d].tableToTopic[table]); // existing
}

// What we should do for additional document (fictional)
private void sweepForAdditionalDocument(int d, int i)
{
   removeWord(d, i); // remove the word i from the state

   int table = 1;
   if (table == docStates[d].numberOfTables) // new Table
   {
      int topic = 1;
      addWord(d, i, table, topic); // sampling its
   }
   else
      addWord(d, i, table, docStates[d].tableToTopic[table]); // existing
}
```

```
/**
 * Decide at which topic the table should be assigned to
 *
 * @return the index of the topic
 */
private int sampleTopic()
{
   double u, pSum = 0.0;
   int k;
   p = ensureCapacity(p, getNumberOfTopics());
   for (k = 0; k < getNumberOfTopics(); k++)
   {
      pSum += numberOfTablesByTopic[k] * f[k];
      p[k] = pSum;
   }
   pSum += gamma / sizeOfVocabulary;
   p[getNumberOfTopics()] = pSum;
   u = random.nextDouble() * pSum;
   for (k = 0; k <= getNumberOfTopics(); k++)
      if (u < p[k])
         break;
   return k;
}

/**
 * Decide at which table the word should be assigned to
 *
 * @param docID
 *           the index of the document of the current word
 * @param i
 *           the index of the current word
 * @return the index of the table
 */
private int sampleTable(int docID, int i)
{
   int k, j;
   double pSum = 0.0, vb = sizeOfVocabulary * beta, fNew, u;
   DOCState docState = docStates[docID];
   f = ensureCapacity(f, getNumberOfTopics());
   p = ensureCapacity(p, docState.numberOfTables);
   fNew = gamma / sizeOfVocabulary;
   for (k = 0; k < getNumberOfTopics(); k++)
   {
      f[k] = (wordCountByTopicAndTerm[k][docState.words[i].termIndex] + beta)
            / (wordCountByTopic[k] + vb);
      fNew += numberOfTablesByTopic[k] * f[k];
   }
   for (j = 0; j < docState.numberOfTables; j++)
   {
      if (docState.wordCountByTable[j] > 0)
```

```
         pSum += docState.wordCountByTable[j]
               * f[docState.tableToTopic[j]];
      p[j] = pSum;
   }
   pSum += alpha * fNew / (totalNumberOfTables + gamma); // Probability for
                                          // t = tNew
   p[docState.numberOfTables] = pSum;
   u = random.nextDouble() * pSum;
   for (j = 0; j <= docState.numberOfTables; j++)
      if (u < p[j])
         break; // decided which table the word i is assigned to
   return j;
}

/**
 * Method to call for fitting the model.
 *
 * @param doShuffle
 * @param shuffleLag
 * @param maxIter
 *           number of iterations to run
 * @param saveLag
 *           save interval
 * @param wordAssignmentsPrintWriter
 *           {@link WordAssignmentsPrintWriter}
 * @param topicsPrintWriter
 *           {@link TopicsPrintWriter}
 * @throws IOException
 */
private void run(int shuffleLag, int maxIter, PrintWriter log)
{
   for (int iter = 0; iter < maxIter; iter++)
   {
      if ((shuffleLag > 0) && (iter > 0) && (iter % shuffleLag == 0))
         doShuffle();
      nextGibbsSweep();
      if (iter % 10 == 0)
         log.println("iter = " + iter + " #topics = "
               + getNumberOfTopics() + ", #tables = "
               + totalNumberOfTables);
   }
}

/**
 * Removes a word from the bookkeeping
 *
 * @param docID
 *           the id of the document the word belongs to
 * @param i
 *           the index of the word
```

```
 */
private void removeWord(int docID, int i)
{
   DOCState docState = docStates[docID];
   int table = docState.words[i].tableAssignment;
   int k = docState.tableToTopic[table];
   docState.wordCountByTable[table]--;
   wordCountByTopic[k]--;
   wordCountByTopicAndTerm[k][docState.words[i].termIndex]--;
   if (docState.wordCountByTable[table] == 0)
   { // table is removed
      totalNumberOfTables--;
      numberOfTablesByTopic[k]--;
      docState.tableToTopic[table]--;
   }
}

/**
 * Add a word to the bookkeeping
 *
 * @param docID
 *           docID the id of the document the word belongs to
 * @param i
 *           the index of the word
 * @param table
 *           the table to which the word is assigned to
 * @param topic
 *           the topic to which the word is assigned to
 */
private void addWord(int docID, int i, int table, int k)
{
   DOCState docState = docStates[docID];
   docState.words[i].tableAssignment = table;
   docState.wordCountByTable[table]++;
   wordCountByTopic[k]++;
   wordCountByTopicAndTerm[k][docState.words[i].termIndex]++;
   if (docState.wordCountByTable[table] == 1)
   { // a new table is created
      docState.numberOfTables++;
      docState.tableToTopic[table] = k;
      totalNumberOfTables++;
      numberOfTablesByTopic[k]++;
      docState.tableToTopic = ensureCapacity(docState.tableToTopic,
            docState.numberOfTables);
      docState.wordCountByTable = ensureCapacity(
            docState.wordCountByTable, docState.numberOfTables);
      if (k == numberOfTopics)
      { // a new topic is created
         numberOfTopics++;
         numberOfTablesByTopic = ensureCapacity(numberOfTablesByTopic,
```

```
                numberOfTopics);
            wordCountByTopic = ensureCapacity(wordCountByTopic,
                numberOfTopics);
            wordCountByTopicAndTerm = add(wordCountByTopicAndTerm,
                new int[sizeOfVocabulary], numberOfTopics);
        }
    }
}

/**
 * Removes topics from the bookkeeping that have no words assigned to
 */
private void defragment()
{
    int[] kOldToKNew = new int[getNumberOfTopics()];
    int k, newNumberOfTopics = 0;
    for (k = 0; k < getNumberOfTopics(); k++)
    {
        if (wordCountByTopic[k] > 0)
        {
            kOldToKNew[k] = newNumberOfTopics;
            swap(wordCountByTopic, newNumberOfTopics, k);
            swap(numberOfTablesByTopic, newNumberOfTopics, k);
            swap(wordCountByTopicAndTerm, newNumberOfTopics, k);
            newNumberOfTopics++;
        }
    }
    numberOfTopics = newNumberOfTopics;
    for (int j = 0; j < docStates.length; j++)
        docStates[j].defragment(kOldToKNew);
}

/**
 * Permute the ordering of documents and words in the bookkeeping
 */
private void doShuffle()
{
    List<DOCState> h = Arrays.asList(docStates);
    Collections.shuffle(h);
    docStates = h.toArray(new DOCState[h.size()]);
    for (int j = 0; j < docStates.length; j++)
    {
        List<WordState> h2 = Arrays.asList(docStates[j].words);
        Collections.shuffle(h2);
        docStates[j].words = h2.toArray(new WordState[h2.size()]);
    }
}

private static void swap(int[] arr, int arg1, int arg2)
{
```

```
   int t = arr[arg1];
   arr[arg1] = arr[arg2];
   arr[arg2] = t;
}

private static void swap(int[][] arr, int arg1, int arg2)
{
   int[] t = arr[arg1];
   arr[arg1] = arr[arg2];
   arr[arg2] = t;
}

private static double[] ensureCapacity(double[] arr, int min)
{
   int length = arr.length;
   if (min < length)
      return arr;
   double[] arr2 = new double[min * 2];
   for (int i = 0; i < length; i++)
      arr2[i] = arr[i];
   return arr2;
}

private static int[] ensureCapacity(int[] arr, int min)
{
   int length = arr.length;
   if (min < length)
      return arr;
   int[] arr2 = new int[min * 2];
   for (int i = 0; i < length; i++)
      arr2[i] = arr[i];
   return arr2;
}

private static int[][] add(int[][] arr, int[] newElement, int index)
{
   int length = arr.length;
   if (length <= index)
   {
      int[][] arr2 = new int[index * 2][];
      for (int i = 0; i < length; i++)
         arr2[i] = arr[i];
      arr = arr2;
   }
   arr[index] = newElement;
   return arr;
}

public int getNumberOfTopics()
{
```

```
   return numberOfTopics;
}

private class DOCState implements Serializable
{

   private static final long serialVersionUID = 1L;
   public int docID, documentLength, numberOfTables;
   public int[] tableToTopic;
   public int[] wordCountByTable;
   public WordState[] words;

   public DOCState(int[] instance, int docID)
   {
      this.docID = docID;
      numberOfTables = 0;
      documentLength = instance.length;
      words = new WordState[documentLength];
      wordCountByTable = new int[2];
      tableToTopic = new int[2];
      for (int position = 0; position < documentLength; position++)
         words[position] = new WordState(instance[position], -1);
   }

   public void defragment(int[] kOldToKNew)
   {
      int[] tOldToTNew = new int[numberOfTables];
      int t, newNumberOfTables = 0;
      for (t = 0; t < numberOfTables; t++)
      {
         if (wordCountByTable[t] > 0)
         {
            tOldToTNew[t] = newNumberOfTables;
            tableToTopic[newNumberOfTables] = kOldToKNew[tableToTopic[t]];
            swap(wordCountByTable, newNumberOfTables, t);
            newNumberOfTables++;
         }
         else
            tableToTopic[t] = -1;
      }
      numberOfTables = newNumberOfTables;
      for (int i = 0; i < documentLength; i++)
         words[i].tableAssignment = tOldToTNew[words[i].tableAssignment];
   }

   public int resolveTopic()
   {
      int[] topicProp = new int[getNumberOfTopics()];
      for (WordState w : words)
      {
```

```
            int table = w.tableAssignment;
            topicProp[tableToTopic[table]]++;
         }
         return ProbUtils.sampleFromProportions(topicProp, random);
      }
   }

   private class WordState implements Serializable
   {
      private static final long serialVersionUID = 1L;
      public int termIndex;
      public int tableAssignment;

      public WordState(int wordIndex, int tableAssignment)
      {
         this.termIndex = wordIndex;
         this.tableAssignment = tableAssignment;
      }
   }

   public List<String> getTopTopicWords(int topic, int top)
   {
      List<String> result = CollectionUtils.newList();
      List<Pair<String, Integer>> words = CollectionUtils.newList();

      for (int i = 0; i < sizeOfVocabulary; i++)
         words.add(new Pair<String, Integer>(codesToWords.get(i),
               wordCountByTopicAndTerm[topic][i]));

      Collections.sort(words, new Comparator<Pair<String, Integer>>()
      {
         public int compare(Pair<String, Integer> a, Pair<String, Integer> b)
         {
            return b.second.compareTo(a.second);
         }
      });

      for (int i = 0; i < Math.min(top, words.size()); i++)
      {
         result.add(words.get(i).first);
      }

      return result;
   }

   public void removeAdditional()
   {
      for (int d = 0; d < docStates.length; d++)
         if (isDocumentAdditional(d))
            for (int i = 0; i < docStates[d].words.length; i++)
```

```
            removeWord(d, i);
   }
}
```

```
package net.msusevastopol.math.ypys.hdp;

import java.util.List;

public interface ICorpus extends IData
{
   public List<IDocument> getDocuments();

   public void addAdditionalDocument(IDocument document);

   public void clearAdditionalDocuments();

   public double printUniqueAuthorsProportionForTopic(Integer topic);
}
```

```
package net.msusevastopol.math.ypys.hdp;

import java.io.FileNotFoundException;
import java.io.IOException;

public interface IData
{
   public void save(String filename) throws IOException, FileNotFoundException;

   public void load(String filename) throws IOException, FileNotFoundException;
}
```

```
package net.msusevastopol.math.ypys.hdp;

import net.msusevastopol.math.ypys.utils.DateUtils;

public interface IDate
{
   public static final IDate ZERO = DateUtils.fromCode(0);

   public int getDay();

   public int getHour();

   public String toShortString();
}
```

```
package net.msusevastopol.math.ypys.hdp;
```

```
public interface IDocument
{
   public String[] getWords();

   public Integer getTopic();

   public void setTopic(Integer topic);

   public IDate getDate();

   public boolean isAdditional();

   public String getAuthor();
}
```

```
package net.msusevastopol.math.ypys.hdp;

public interface IPeakDetector
{
   public IDate[] detectPeaks(FrequencyOverTime corpus);
}
```

```
package net.msusevastopol.math.ypys.hdp;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.Arrays;
import java.util.List;

public class Main
{
   private static final int WORDS_IN_ADD_DOC = 50;
   private static final int KEYWORDS_TO_PRINT = 50;
   private static final int ADD_DOCS = 10000;
   private static final int TOP_COUNT = 50;
   private static final int TARGET_TOPIC = 1;
   private static final int FILENAME_TOP_COUNT = 50;
   private static final int HDP_ITERATIONS = 300;
   private static final double SD2_TRESHOLD = 3.0;
   private static final double DOT_PROD_TRESHOLD = 0.95;

   public static void main(String[] args) throws IOException
   {
      PrintWriter log = new PrintWriter(System.out, true);
      ICorpus corpus = new Corpus(Filename.INPUT_EX);
      FrequencyOverTime freq = new FrequencyOverTime(corpus);
      NedModel ned = new NedModel(corpus, log);

      ICorpus nedResult = ned.run();
```

```
nedResult.save(Filename.NED_FILE);


int n = 30;
IDate[] result = new MaxPeakDetector(n, 9).detectPeaks(freq);
int index = 1;
FrequencyOverTime[] results = new FrequencyOverTime[n];
for (IDate date : result)
{
   log.println("Found peak #" + index + ": " + date);
   FrequencyOverWords keywords = FrequencyOverWords.getFromCorpus(
         corpus, null, date, 0);

   String[] topWords = keywords.getTopWords(FILENAME_TOP_COUNT);
   log.println("Keywords for the peak: " + Arrays.toString(topWords));

   WordProp[] top = keywords.getTopWordsExtended(TOP_COUNT);
   IDocument additionalDocument = Document.getFromProp(top,
         WORDS_IN_ADD_DOC);

   for (int i = 0; i < ADD_DOCS; i++)
      corpus.addAdditionalDocument(additionalDocument);
   HDPGibbsSampler hdp = new HDPGibbsSampler(corpus, log);
   hdp.run(HDP_ITERATIONS);
   String suffix = "" + index;
   corpus.save(Filename.getDocumentsToTopicAssignment(suffix));

   hdp.removeAdditional();

   List<String> topTopicWords = hdp.getTopTopicWords(TARGET_TOPIC,
         KEYWORDS_TO_PRINT);

   log.println("Topic keywords are: " + topTopicWords);

   FrequencyOverTime freqOverTime = new FrequencyOverTime(corpus,
         TARGET_TOPIC);

   freqOverTime.save(Filename.getPlotOutput(suffix + "-"
         + TARGET_TOPIC));

   double sd2 = freqOverTime.getSD2();
   log.println("SD^2: " + sd2);
   if (sd2 > SD2_TRESHOLD)
      log.println("Note: sd > sd_0");

   corpus.clearAdditionalDocuments();

   double authorProp = corpus
         .printUniqueAuthorsProportionForTopic(TARGET_TOPIC);
   log.println("Unique authors proportion: " + authorProp);
```

```
            results[index - 1] = freqOverTime;

            index++;
         }

         for (int i = 0; i < results.length; i++)
         {
            for (int j = 0; j < i; j++)
            {
               double value = results[i].normalizedDotProduct(results[j]);
               log.println(i + " " + j + " " + value);
               if (value > DOT_PROD_TRESHOLD)
                  log.println("Note: events " + i + " and " + j
                        + " are simular");
            }
         }
      }

      private static class Filename
      {
         // Prevent instantiation.
         private Filename()
         {
         }

         private static String getPlotOutput(String suffix)
         {
            return append(PLOT_OUTPUT, suffix);
         }

         private static String getDocumentsToTopicAssignment(String suffix)
         {
            return append(DOCUMENT_TO_TOPIC_ASSIGNMENT, suffix);
         }

         private static String append(String filename, String suffix)
         {
            return filename + "-" + suffix + ".txt";
         }

         private static final String INPUT_EX = "data/inputEx.txt";
         private static final String PLOT_OUTPUT = "data/out/plotOutput";
         private static final String DOCUMENT_TO_TOPIC_ASSIGNMENT =
             "data/out/DocumentToTopicAssignmentEx";
         public static final String NED_FILE = "data/out/NedAssignment.txt";
         public static final String TMP_CORPUS = "data/out/TmpAssignment.txt";
         public static final String TMP_CORPUS2 = "data/out/TmpAssignment2.txt";
      }
   }
```

```
package net.msusevastopol.math.ypys.hdp;

import java.util.Collections;
import java.util.Comparator;
import java.util.List;
import java.util.Map;

import net.msusevastopol.math.ypys.utils.CollectionUtils;
import net.msusevastopol.math.ypys.utils.DateUtils;

public class MaxPeakDetector implements IPeakDetector
{
   private int numberOfPeaks;
   private int tresholdRadius;

   public MaxPeakDetector(int numberOfPeaks, int tresholdRadius)
   {
      this.numberOfPeaks = numberOfPeaks;
      this.tresholdRadius = tresholdRadius;
   }

   public IDate[] detectPeaks(FrequencyOverTime freq)
   {
      Map<IDate, Integer> mapping = freq.getMapping();
      List<Pair<IDate, Integer>> list = CollectionUtils.newList();
      for (IDate date : mapping.keySet())
         list.add(new Pair<IDate, Integer>(date, mapping.get(date)));

      Collections.sort(list, new Comparator<Pair<IDate, Integer>>()
      {
         public int compare(Pair<IDate, Integer> a, Pair<IDate, Integer> b)
         {
            return b.second - a.second;
         }
      });

      List<IDate> result = CollectionUtils.newList();
      for (Pair<IDate, Integer> pair : list)
      {
         if (result.size() >= numberOfPeaks)
            break;

         IDate date = pair.first;
         boolean add = true;
         for (IDate rdate : result)
         {
            if (DateUtils.diffHours(date, rdate) <= tresholdRadius)
            {
               add = false;
               break;
```

```
            }
         }

         if (add)
            result.add(date);
      }

      return result.toArray(new IDate[0]);
   }
}
```

```
package net.msusevastopol.math.ypys.utils;

import net.msusevastopol.math.ypys.hdp.WordProp;

public class MiscUtils
{
   // Prevent instantiating.
   private MiscUtils()
   {
   }

   public static String implode(String separator, String... elements)
   {
      StringBuilder sb = new StringBuilder();
      if (elements.length > 0)
      {
         sb.append(elements[0]);
         for (int i = 1; i < elements.length; i++)
            sb.append(" ").append(elements[i]);
         return sb.toString();
      }
      return "";
   }

   public static void normalize(WordProp[] prop)
   {
      double sum = 0.0;
      for (WordProp wp : prop)
         sum += wp.prop;
      for (WordProp wp : prop)
         wp.prop /= sum;
   }
}
```

```
package net.msusevastopol.math.ypys.hdp;

import java.io.PrintWriter;
import java.util.Map;
```

```
import java.util.Set;

import net.msusevastopol.math.ypys.utils.CollectionUtils;

public class NedModel
{
   private PrintWriter log;

   private static int N_LAST = 10000;
   private static double TRESHOLD = 0.95;

   private int[][] documents;
   private int vocabularySize;
   private ICorpus corpus;

   private Map<Integer, String> codeToWord = CollectionUtils.newMap();
   private Map<String, Integer> wordToCode = CollectionUtils.newMap();

   private int nLast;
   private double treshold;

   public NedModel(ICorpus corpus, PrintWriter log, int nLast, double treshold)
   {
      this.log = log;
      this.nLast = nLast;
      this.treshold = treshold;
      this.corpus = corpus;

      documents = new int[corpus.getDocuments().size()][];
      vocabularySize = 0;
      for (int d = 0; d < documents.length; d++)
      {
         int documentLength = corpus.getDocuments().get(d).getWords().length;
         documents[d] = new int[documentLength];
         for (int w = 0; w < documentLength; w++)
         {
            String word = corpus.getDocuments().get(d).getWords()[w];
            if (!wordToCode.containsKey(word))
            {
               wordToCode.put(word, vocabularySize);
               codeToWord.put(vocabularySize, word);
               vocabularySize++;
            }
            documents[d][w] = wordToCode.get(word);
         }
      }
   }

   public NedModel(ICorpus corpus, PrintWriter log)
   {
```

```
   this(corpus, log, N_LAST, TRESHOLD);
}

public ICorpus run()
{
   ICorpus result = new Corpus();
   int[] df = new int[vocabularySize];
   Map<Integer, Double>[] weights = new Map[documents.length];
   for (int d = 0; d < documents.length; d++)
   {
      Map<Integer, Double> weight = CollectionUtils.newMap();

      Set<Integer> words = CollectionUtils.newSet();
      for (int word : documents[d])
         words.add(word);

      double z = 0;
      for (int word : words)
      {
         int tf = 0;
         for (int w1 = 0; w1 < documents[d].length; w1++)
            if (word == documents[d][w1])
               tf++;

         df[word]++;

         double tfidf = tf * Math.log((d + 2) / df[word]);
         weight.put(word, tfidf);
         z += tfidf * tfidf;
      }

      z = Math.sqrt(z);
      for (int word : weight.keySet())
         weight.put(word, weight.get(word) / z);

      weights[d] = weight;

      if (d < nLast || documents[d].length <= 3)
         continue;
      boolean isNew = true;

      for (int q = d - nLast; q < d; q++)
      {
         double dotProd = 0;
         Set<Integer> sharedWords = CollectionUtils.newSet();
         for (int w : documents[d])
            sharedWords.add(w);
         for (int w : documents[q])
            sharedWords.add(w);
```

```
            for (Integer word : sharedWords)
            {
               double dv = CollectionUtils.getDefault(weights[d], word,
                     0.0);
               double qv = CollectionUtils.getDefault(weights[q], word,
                     0.0);

               dotProd += dv * qv;
            }

            if (1 - dotProd < treshold)
            {
               isNew = false;
               break;
            }
         }

         if (isNew)
         {
            result.getDocuments().add(corpus.getDocuments().get(d));
         }

         if (d % 100 == 0)
            log.println(d + " : " + result.getDocuments().size());
      }

      return result;
   }
}
```

```
package net.msusevastopol.math.ypys.hdp;

public class Pair<K, V>
{
   public K first;
   public V second;

   public Pair(K first, V second)
   {
      this.first = first;
      this.second = second;
   }
}
```

```
package net.msusevastopol.math.ypys.utils;

import java.util.Random;

public class ProbUtils
```

```
{
   // Prevent instantiating.
   private ProbUtils()
   {
   }

   public static int sampleFromProportions(int[] prop, Random r)
   {
      if (prop == null || prop.length == 0)
         throw new IllegalArgumentException(
               "Array must be not-null and its size must be greater than zero");

      int sample = r.nextInt(sum(prop));
      for (int i = 0, temp = 0; i < prop.length; i++)
      {
         temp += prop[i];
         if (temp > sample)
            return i;
      }

      throw new IllegalStateException("This will not happen");
   }

   private static int sum(int[] prop)
   {
      int sum = 0;
      for (int i : prop)
         sum += i;
      return sum;
   }
}
```

```
package net.msusevastopol.math.ypys.hdp;

public class WordProp
{
   public String word;
   public Double prop;

   public WordProp(String word, Double prop)
   {
      this.word = word;
      this.prop = prop;
   }
}
```
