Project Historian: Automatic Timeline Generation

A tool to bring context to news stories

Tiancheng Zhang
Class of 2018
Advisor:
Prof. Susan E. McGregor

Abstract

This paper describes Project Historian, a software system that combines various natural language processing techniques to automatically generate news event timelines on-demand from a news story archive. It is designed for working journalists with the goal of making it easy to add context to upcoming and evolving news stories.

Keywords: natural language processing, newsroom automation, unsupervised learning, topic detection

Contents

Chapter 1.

Introduction

Journalism often requires reporters to shift to new beats, or cover big events as they unfold, yet to impress on the reader why the event matters, sufficient context needs to be presented as part of the package. However, researching this context is often time-consuming and difficult, and few publications have a system in place to assist journalists on deadline. Likewise, readers do not have the time to trawl through weeks of archives looking for background information.

This is where Project Historian comes in. It is a tool aimed at journalists that automatically creates a timeline of related events from news archives, based on a user's search query. The timeline can then be used as background research for the journalist as they write a news story, or even edited and embedded directly in a story for the reader's reference.

1.1. Providing Context to News Stories

Project Historian serves the dual purpose of saving time for journalists and making news coverage more relevant to the public. Because news events often evolve over long spans of time, it can be hard for all but the most dedicated news readers to follow a particular story, such as the current administration's back-and-forth on DACA, or the Russia investigation. In many cases, the information needed to provide context for each new story is already available in a publication's archive, but it can be time-consuming for the journalist to compile them for the reader. Project Historian aims to speed up this process, making it easier for the journalist to provide more context for stories.

1.2. Timelines in News Coverage

Current tools for creating news timelines all require substantial time investment on the journalist's side to research the content and manually enter it into a particular data format. Project Historian automates this process using natural language processing technologies, allowing journalists conducting background research to use news archives more selectively. In addition, the easy timeline creation process makes it more feasible for journalists to use news timelines more frequently to augment the context of individual news stories.

Chapter 2.

Background Review

While journalism-focused timeline creation tools exist, few natural language processing algorithms are designed with journalists in mind – and no current tool combines the two technologies efficiently. The following sections discuss some of the currently existing tools that serve as reference points to Project Historian in detail.

2.1. Natural Language Processing

The key technical challenge in Project Historian is effective detection of what a news story in a publication's archive is about, or its "gist". Once the main elements of a story have been identified, it can be added to an event cluster that represents coverage around a certain facet of the story. For example, a group of stories that all touch on the charges against Paul Manafort in the Mueller investigation might be grouped into a cluster on that basis.

2.1.1. TF-IDF

One of the simplest ways of determining the subject of a story is to look for prominent and/or repeated words. For example, the TF-IDF metric weights a word based on how frequently a word appears in a given document as compared to others in the corpus. Although relatively simple, this method has proven highly effective at retrieving highly relevant documents for a query [1].

2.1.2. Dependency parsing

Dependency parsers makes it possible to extract the “who-did-what” structures from sentences. For example, a dependency parse of “Trump signals support for tougher background checks” will identify “signal” as the root verb in the sentence, with “Trump” being the subject and “support for tougher background checks” being the object. A classification tool can then, in the next step, perform the clustering using the subjects, verbs and objects in each story identified by the dependency parser. In this case, having “Trump” as the subject shows that the story will be about President Trump. A widely used library that gives this capability is the Stanford `CoreNLP` library [2], which helpfully also includes a tool for entity recognition [3], a family of research that promises to extract named entities such as organizations and names from texts.

While this method works in theory, in practice the parse tree of a long sentence can be complex and yield long phrases of doubtful utility. Even “support for tougher background checks”, which is not particularly complex, is difficult to match in a corpus as each story will likely phrase the concept differently.

2.1.3. Topic modeling

Another family of methods, referred to as topic modeling, categorizes a collection of articles into different “topics” and assigns different weights to each topic for every article, where the meaning of each topic is hopefully captured by a list of words that the algorithm identifies as “topic words” from the articles. Some examples of topic modeling algorithms include Latent Dirichlet Allocation [4], which uses a probabilistic model, and Latent Semantic Indexing [5], which first creates a matrix of terms that appear in a document, and then reduces the dimensionality of the matrix to produce topics.

2.1.4. Neural networks

For the above methods, a key challenge is that factors such as morphological forms (single and plural forms or adjective and adverb forms of a given word, etc.) as well as synonyms may not resolve well. For example, a news might use the word

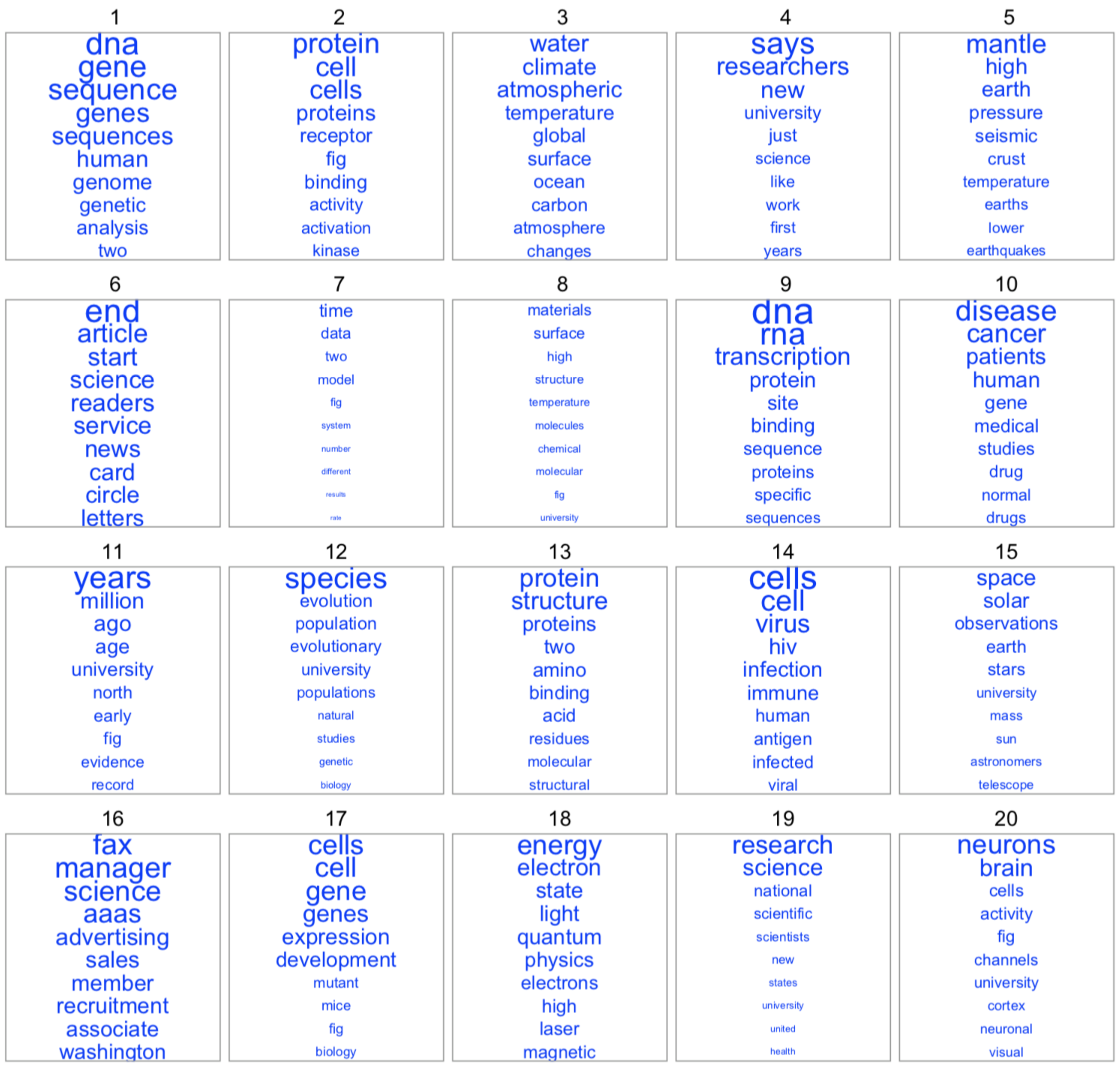


Figure 2.1.: A visualization of topics from a dataset consisting of PNAS, *Science* and *Nature* articles from 1880 to 2005, where each topic is a list of words. Words with larger font sizes are more important in a topic. LDA is used here. [5]

"Kremlin" as a shorthand for Russia; an ideal analytical tool should understand the interchangeability between the two words.

This is where a fourth family of natural language processing methods, known as "neural networks", comes in. A neural network is a family of machine learning architecture that creates a vectorized representation of words within a body of text. Referred to as word embeddings, these vectors express the meaning of words in a text as a set of numeric values, or vectors and have been shown to capture word meanings very effectively [6]. For example, the vectorized distance between "king" and "man" in word embedding is roughly equivalent to the distance between "queen" and “woman” [7]. Such a model has the potential to solve the word meaning linkage issue such as the Russia/Kremlin equivalence described above.

Early work using word embeddings for journalistic purposes has shown tremendous promise. For example, Roberts of Austin Cut Research Group used word embeddings to detect topic words in automatically transcribed political speeches, resulting in highly thematically coherent topic word clusters [8]. Word embedding models are thus a key component of Project Historian.

Lastly, commercial natural language processing services, such as Google’s natural language API, are worth mentioning. Although they do perform well, the prices they charge are out of reach for the free and open tool that Project Historian aims to be.

2.2. Visualization

Visualization is an important aspect of Project Historian, because it offers an efficient and usable method of presenting the potentially large number of stories related to a topic. Fundamentally, Project Historian aims to provide a digest of past coverage on a topic; the point would be moot if a reader still has to plow through a list of stories. Because more recent coverage may be more relevant to understanding current events, a timeline representation was chosen over other visual forms, such as network diagrams. Moreover, timelines are better understood by general readers, make Project Historian’s dual-use case more feasible.

A broad range of timeline creation tools can be found on the web. They can be broadly categorized into two categories: input-based and code-based.

2.2.1. Input-based timeline creation

This type of timeline creation tools requires the user to input the content that goes into the timeline by either filling out a web form or uploading a spreadsheet with the timeline content; the latter are frameworks that can be used to programmatically generate timelines, usually with customization options. For Project Historian, the former can serve as design reference points, while the latter can be used to build the visualization component.

Timeline JS - timelines for journalism

A good example of an input-based tool is the Timeline JS tool made by Northwestern University's Knight Lab [9], which is explicitly aimed at journalists. Timelines made by the tool have appeared in *CNN* [10], the *TIME Magazine* [11] and *Engadget* [12], among others, according to the tool's website. The tool creates elegant slideshow-like timelines, with a large content window and a scrolling time axis below it. To use it, a journalist must put the elements/events into a spreadsheet manually and then upload it to the tool. This requirement means significant time goes into the creation of timelines, as the journalist must research and pick which story to appear in the timeline. Perhaps as a natural result, the featured stories on the Timeline JS page either do not cover a long time span, or appear in stories that are not time-sensitive. It's graphic-intensive style, however, is a good design cue that Project Historian can take.

Additional timeline tools

The other input-based timeline tools, such as TimeGraphics [13], Vizzlo [14] and timeglider [15], work in similar ways, but are not targeted at journalism, which means that their support for important story elements, such as images, hyperlinks and social media posts, is limited. However, Project Historian can certainly borrow design ideas from these tools.

2.2.2. Code-based timeline creation

One existing code-based timeline tool that looks particularly promising for Project Historian is `vis.js`, a Javascript-based library that supports both programmatic timeline creation and has a built-in GUI editing interface [16].

Chapter 3.

Target Audience

Project Historian is mainly designed for the benefit of working journalists. Online publications, especially those that cover breaking news, will benefit in particular. Researchers and scholars may also find Project Historian useful for organizing and exploring research materials.

3.1. Journalists

Project Historian can enhance journalists' work in both reporting and presentation. For reporting, the timelines produced by Project Historian can serve as useful background study for a journalist embarking on a new story or subject area. By looking over past events related to a topic, journalist can better understand the motives and stakeholders of a story thread, identify sources to reach out to or discover new story ideas based on what is missing from existing reports. In this sense, Project Historian function as something like a digital "clip file" that is automatically generated from a publication's archives.

For publishing, the timeline visualization produced by Project Historian can present the context of news stories concisely when embedded in articles. The reader can skim the headlines in the timeline to understand the background of an event and click through the story links for more detailed reporting.

3.2. Publications

Project Historian can benefit publications by extending the lifespan of its story archive, resurfacing important stories (such as longform investigative pieces that broke a story) when follow-up news break, which can bring new traffic and traction to stories that the publication had once spent significant resources on.

Sometimes, breaking events can bring an older story back into the public's attention because it provides important background information on the thread of events. One such example is a video explainer of the Syrian Civil War published by *Vox* in October 2015, which saw a traffic spike a month after its publication when the Paris terrorist attack made people interested about the conflicts in Syria [17]. Project Historian can help publications resurface important previous coverage in a similar manner, either by an editor identifying which stories are important in a timeline leading up to a breaking event, or by including a timeline in a breaking story and promoting stories that received a significant traffic boost from the timeline.

3.3. Scholars

In addition to journalists, Project Historian may be of interest to scholars and educators studying event timelines and wishing to present or analyze the media narratives around them. For example, Project Historian can be used to learn if U.S. media organizations are covering the Iran nuclear issue versus the North Korean nuclear issues differently. A scholar interested in learning the political leanings of different publications can look at the different timelines produced by stories from the publications of interest, or what opinion articles are published on different publications for the same timeline topic. Skimming through the stories in the timelines saves one time digging through archives looking for relevant stories, and might prompt questions for further investigation.

Chapter 4.

Technology Overview

4.1. Data

4.1.1. RSS

Project Historian currently uses the freely-available RSS feeds of reputable publications, including *CNN*, *New York Times*, *Reuters*, *USA Today*, *Vox*, *Washington Post* and *Wall Street Journal* as the source of its corpus. The reason is that these RSS feeds are official, publicly available and up-to-date. Users can easily expand or modify the list of feeds by editing a text file containing the list of URLs to target.

The sources RSS feeds are downloaded via the feedly Streams API, a RESTful API that greatly simplifies the process of retrieving RSS feed updates without altering the feed content. New RSS feeds can be added as simply as adding the URL of the desired RSS feed to an existing list of RSS feeds.

For the Project Historian prototype, story caching began in November 2017, although there might have been interruptions since the caching began due to the system being powered off or offline during the scheduled caching time.

4.1.2. Internal integration with a news publication

A news publication can easily configure Project Historian to use its own archive as data source by pulling its archive into the database format that Project Historian uses and configuring Project Historian to use this database instead of the RSS-based database, which enables Project Historian to rely on the archive of the specific publication. The database format will be included in the appendix.

4.1.3. SQLite

The cached feeds are stored in a SQLite [18] database on a web server that also hosts Project Historian's web interface. While each user can run their own server and perform their own data caching, Project Historian will also have a public server for occasional users. SQLite is chosen over more sophisticated database solutions because of its accessibility: the database can be easily moved, no SQL server setup is necessary and Python has built-in support for interfacing with it. One of Project Historian's goals is accessibility, meaning that ideally, a journalist will be able to set it up and use it without much experience or difficulty. Relying on SQLite goes a long way to achieving that goal.

4.2. Natural Language Processing

4.2.1. Natural Language Toolkit

The Natural Langauge Toolkit (`NLTK`) [19] is a Python toolkit that provides a comprehensive list of natural language processing functions, and is the tool of choice for basic text processing in Python (a few project examples include [20] and [21]). In Project Historian, its use is limited to basic rule-based sentence and word tokenization (the breaking down of text into words, sentences and phrases) as well as providing a list of common English words (stopwords) to ignore, such as articles and prepositions. Text from RSS feeds are first tokenized using `NLTK` before further processing.

4.2.2. `gensim`

The `gensim` [22] package is a Python library that provides a variety of unsupervised semantic modeling methods. In Project Historian, an implementation of Google word2vec's phrase detection ("word2phrase") [23] is used to detect phrases by considering the frequency of two words occurring together versus separately; if two words appear together often but seldom appear separately, they are likely to belong to a phrase.

`gensim` is chosen for this task because it provides a Python implementation of the word2phrase algorithm. Phrase detection is performed on the tokenized RSS text prior to training a word embedding model.

4.2.3. `FastText`

`FastText` [24, 25]is the neural network component in Project Historian. It is a high-performing toolkit for training a word embeddings model.

In addition to its high performance in terms of both training time and model effectiveness, it also utilizes sub-word n-grams [1] in its training, which is especially helpful in journalism where "new" combination words, such as "trumpcare", are being coined all the time. In Project Historian, `FastText` is used to train a word embedding model on the tokenized and then "phrased" RSS text, in order to identify both the content of stories and the similarity level between stories.

4.2.4. `scikit-learn`

The `scikit-learn` [26]package is a Python library that provides a wide array of machine learning functions that are high-performing and easy to use, making it an ideal choice for Project Historian. The main role of `scikit-learn` in this project is giving TF-IDF value for words in a piece of text, providing a distance metric between words and documents and providing a clustering algorithm for vectorized presentations of stories. The TF-IDF values are then used as weights to obtain a weighted average of word vectors in a story as the vector representation of that story, which is used to cluster individual stories together for presentation in the Project Historian interface.

[1] An n-gram refers to a sequence of n characters. For example, there are two tri-grams in the word "text", "tex" and "ext".

4.3. Front-End Presentation

This section describes the web-based tools used to present the interactive results display of Project Historian.

4.3.1. `Flask`

`Flask` [27] is a lightweight Python web development framework. It is used to serve up Project Historian's web interface for its simplicity in both development and deployment.

4.3.2. `Materialize`

`Materialize` [28] is a CSS framework that implements Google's Material design. It is chosen as the style framework for Project Historian's web front-end because it is easy to develop for and provides a simple, modern interface.

4.3.3. `vis.js`

`vis.js` [16] is a Javascript visualization library that enables developers to create visually-pleasing web-based interactive tools, timelines being one of them. The final visualization of Project Historian is written using `vis.js` and the visualization takes the form of code that can be easily copied and pasted into online stories.

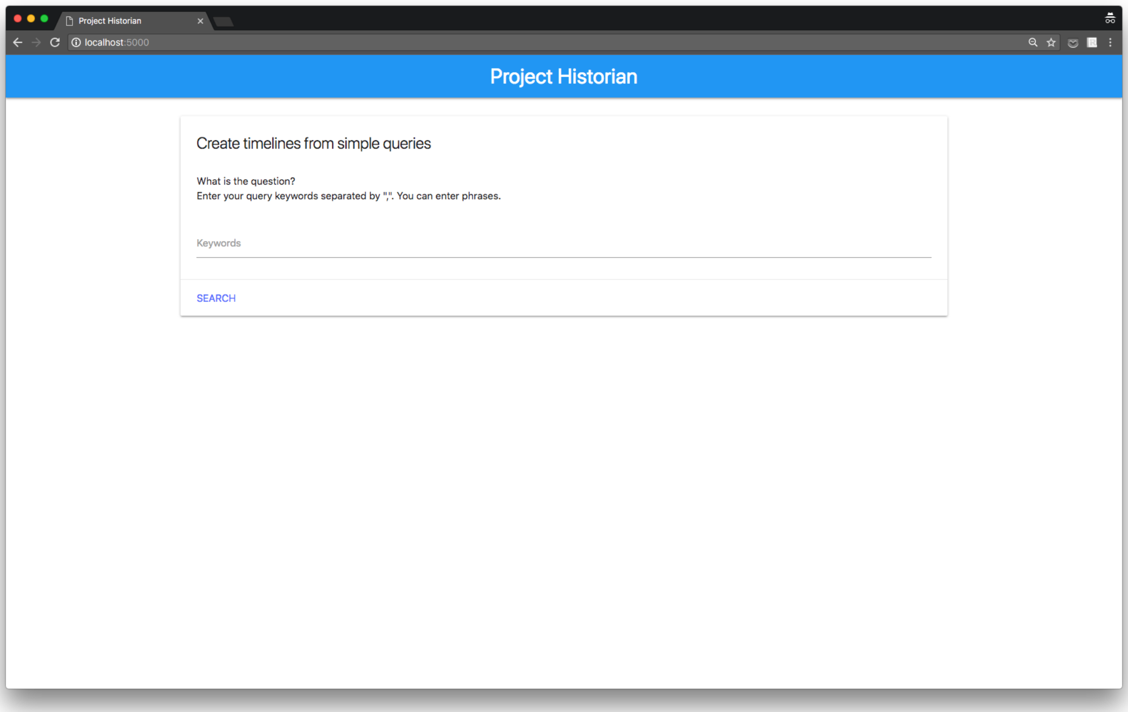

Project Historian
localhost:5000
Project Historian
Create timelines from simple queries
What is the question?
Enter your query keywords separated by ",". You can enter phrases.
Keywords
SEARCH

Figure 4.1.: The interface of Project Historian runs on a server powered by `Flask`. The interface is built using `Materialize`.

Chapter 5.

System Architecture

Project Historian consists of two overall functional units: the offline unit, which handles data collection and model training; and the online unit, which handles user queries on-demand.

5.1. Offline Unit

The offline unit of Project Historian handles the collection of story backlog for analysis, as well as the time-consuming part of the project's required natural language processing. It is updated on a daily basis to incorporate the latest news events into its analysis in a reasonably timely fashion, processing the current volume of data in less than a day.

5.1.1. Backlog collection

In its current form, backlog collection is performed using RSS caching. A list of RSS feeds is queried and cached into a database on a daily schedule. The variety of RSS sources not only ensures that Project Historian covers a reasonable breadth of events, but also makes a large enough corpus for natural language processing techniques to work well. Currently cached RSS feeds include sections of *CNN*, *New York Times*, *USA Today*, *Washington Post*, *Vox* and *Wall Street Journal*.

If Project Historian is eventually adapted to be used internally for a publication, the publication will have the option to switch the data source to its own backlog. A

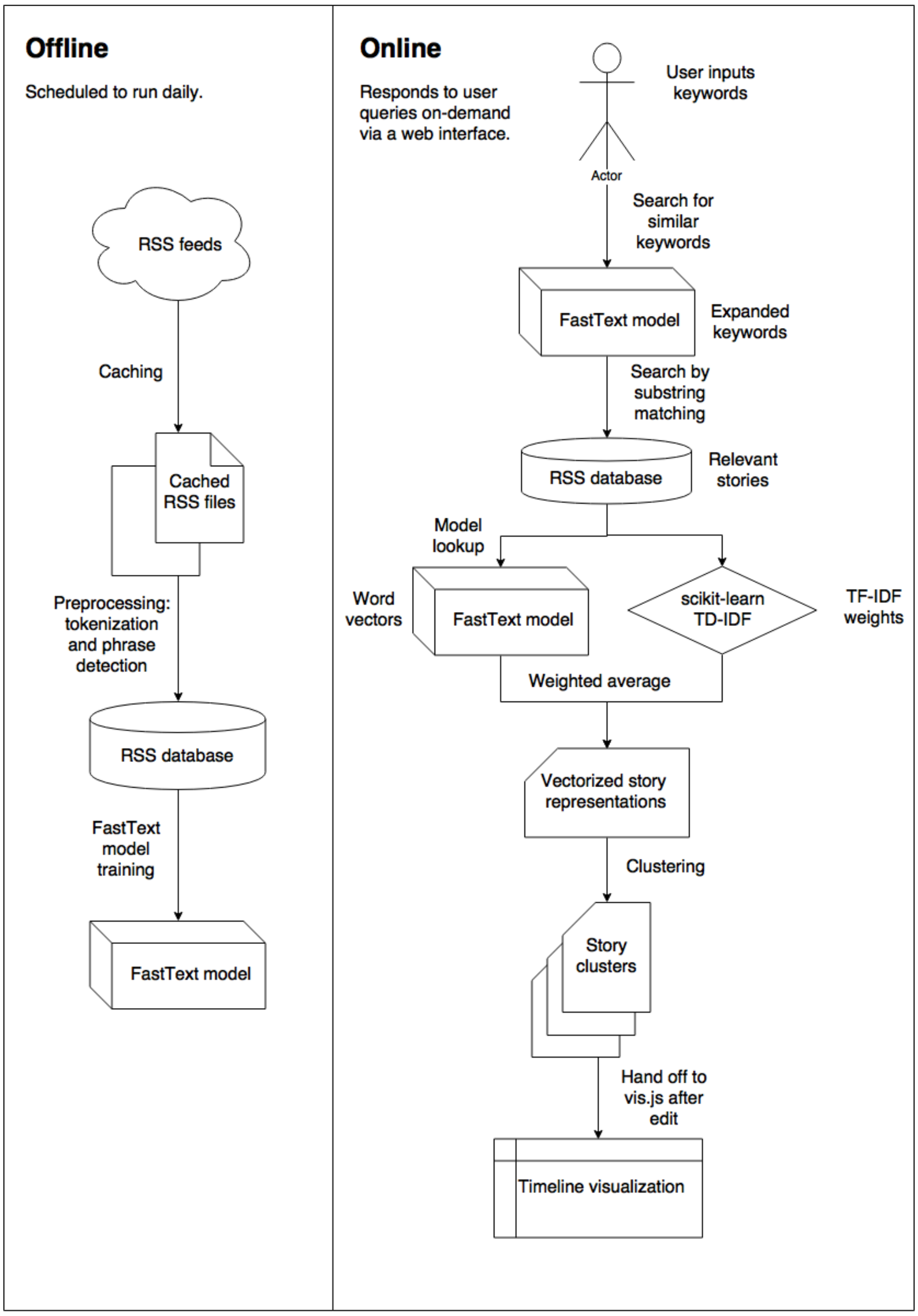


Figure 5.1.: A diagram of Project Historian's structure.

script can be added to convert the publication's backlog into the database format that Project Historian uses, which then enables Project Historian to utilize the publication's backlog for its data source. While this approach would reduce the variety of news sources, it would more than make up for it in the richness of the news corpus for use in the next steps of Project Historian.

5.1.2. Preprocessing

After the caching step, a preprocessing step is run on the full text of each feed item, including the title, the text summary and the text content. The preprocessed text is then stored in the database to be used as the text for clustering later.

The preprocessing step begins with a tokenization step, which splits the text into sentences and then sentences into words. This step is facilitated by the `NLTK` [19] library.

A simple data-driven phrase detection algorithm, described in the work by Mikolov et al. [23], is run over the tokenized corpus twice with different thresholds to detect common word collocations. An initial run with a high threshold aims to find names (e.g. "bob corker", "mike pence") while a second run with a lower threshold aims to find other phrases (e.g. "capitol hill", "white house", "russia investigation") as well as phrases formed from names detected in the previous run (e.g. "senator bob corker"). The phrase detection step uses an implementation of the algorithm in the `gensim` package. The detected phrases are turned into single tokens by replacing spaces with underscores and treating them as distinct words in later steps.

5.1.3. Model training

A skipgram word embedding model, which learns word representations by having a neural network predict a word's context words (i.e. words that appear within a short window around the targeted word), is trained on the entire story backlog on a daily basis, using the `FastText` [24, 25] package, after the caching and preprocessing processes of the day is complete. This word embedding model is later used in event clustering.

While many tried-and-true pre-trained word embeddings, such as `word2vec` embeddings and `GloVe` embeddings, can be easily found on the web, training a custom model is chosen over using a pre-trained word embedding model for the following reasons:

1. **Timeliness.** Pre-trained word embeddings are commonly trained on older data. The Google `word2vec` pre-trained embeddings was released in 2013; the `GloVe` embeddings were released in 2014. While these models are applicable to general NLP applications, news analysis presents unique challenges in terms of a constantly evolving vocabulary of entities. Neither models would have an idea about, for example, how George Papadopoulos is related to Russians, or even how Trump is closely related to the presidency of the United States. Project Historian's `FastText` model, however, captures both quite well.

2. **Domain specificity.** Pre-trained word embeddings are trained on a much broader range of topics than would be necessary for news event clustering. Project Historian does not need to know, for example, the capitol of every country in the world (an example often used to demonstrate the effectiveness of word embeddings). Such lack of specificity would bring with it a much larger vocabulary than needed and as a result, unnecessary performance penalty.

3. **Performance and resource issues.** Project Historian is designed to be run on a web server with limited memory. Larger pre-trained word embedding models, due to their large vocabulary, require a large amount of memory, which can be cost-prohibitive to obtain. Considering that much of this large vocabulary is irrelevant to Project Historian, running a lightweight model trained on a relevant corpus is a better choice.

4. **Corpus quality.** Journalism stories are, in general, hand-edited, high quality writing. Therefore, not only does it capture a large amount of relevant information in a relatively compact corpus (compared to, for example, social media), it also requires much less concern about common issues of generic web-based corpora such as inaccurate information and spelling mistakes. As a result, despite being trained on a much smaller corpus, Project Historian's `FastText` model provides good enough usability, at least for Project Historian's use case.

Beyond this, `FastText` also has the specific advantage of providing word vectors for any word or phrase that is not in the training corpus by using sub-word n-grams, a useful feature for newly-coined terms such as "trumpcare".

In its current implementation, Project Historian takes around 20 minutes each day to train a model that is around 1GB in size.

The model is tuned via its ability to find high-quality related words to a hand-selected set of test words based on recent events. For example, a high-quality model should be able to find that “moscow” and “kremlin” are highly related to “russia”, as journalists often use these words interchangeably to refer to Russia in their new stories.

5.2. Online Unit

The online unit responds to user queries and generates event clusters for the user. With the trained model in place, the rest of the natural language processing happens here, in the following steps.

5.2.1. Query handling

A query starts with a user inputting query keywords – such as recent news topics – into the system. Project Historian first expands the keywords by finding closely related words in the FastText embedding model (using cosine similarity[1]) , and then runs a query in the database by matching the expanded list of keywords in the preprocessed corpus.

For example, if a user inputs “trump”, Project Historian will find closely related words such as “president” and “white house”, and consider an entry a hit if it contains any of the three words/phrases.

If multiple keywords are entered, Project Historian will expand each of them. In the database query, each keyword will be connected with its alternatives using a OR relation, while each group of expanded keywords will be connected with an AND relation. For example, if “Trump” and “Russia” are entered and the system offers up “White House” and “President” for “Trump” and “Kremlin”, “Moscow” for “Russia”, the query will look for stories that contain at least one word among “White

[1]Cosine similarity measures similarity between two vectors by considering the cosine value of the “angle” they form in the vector space. The larger the value, the smaller the angle and the more similar the two vectors are. [29]

House", "President" and "Trump" and any one word among "Kremlin", "Moscow" and "Russia".

5.2.2. Forming story representations

Once the query returns a list of stories, a TF-IDF transformation is run over the tokenized returned corpus to determine the importance of each word/phrase within story. The TF-IDF values are then used as weights for the importance of each word in the text after some transformation, and a weighted average of all word vectors are taken to form a vectorized presentation of the story[2].

While more sophisticated approaches for obtaining salient components of a document exist, such as dependency parsing or named entity recognition (NER), TF-IDF is chosen for the following reasons:

1. **Simplicity and speed.** TF-IDF calculation is much faster than any deeper-level text processing, which is crucial for the online portion of Project Historian as the user will be waiting for the results of their query.
2. **Adaptability.** Many parsing or NER algorithms are based on statistical results or inferred rules from existing corpora, which limits their ability to adapt to new concepts, a particularly common occurrence when dealing with up-to-date news, since statistics or rules suitable for a new name or organization might not be available. TF-IDF, however, does not concern itself with what is inside a token. This lack of insight conversely makes it very adaptable to new entities and concepts.
3. **Quantifiability.** Even if parsing or NER correctly identifies entities in a document, a salience value would still be preferable for each entity discovered. TF-IDF inherently provides such values.

[2]For each story, the TF-IDF values of each token are thresholded and used as weights for calculating the story vector. The rationale behind this approach is to extract the salient information from each story with TF-IDF-based weights.

5.2.3. Story clustering

Once a vectorized representation of each story has been obtained, a clustering algorithm can be run to cluster entities into events.

An adaptive clustering algorithm, based on the K-Means algorithm, is used for this step. The algorithm repeatedly split each cluster until every cluster meets a quality standard. The quality of a cluster is measured by the average distance of each node in a cluster to the cluster's centroid. The pseudocode for the algorithm can be found in the technical appendix.

K-Means, which clusters by iteratively choosing cluster centers based on the previous iteration of clustering results (the centroids of the previous iteration of clusters are used as the base for the next iteration, until the centroids stop changing), is chosen as the base for this algorithm because it provides an intuitive measure of cluster qualities by having a centroid for each cluster to measure distance against.

Chapter 6.

Deployment and Usage

Project Historian is designed to be highly lightweight and portable. This chapter discusses its deployment and usage.

6.1. Deployment

Project Historian is designed with the newsroom in mind. A reporter with a desktop computer can set it up and keep it running if she wants to; a newsroom that finds this tool useful can lease a virtual server from a cloud computing provider or set up a local machine to host it; and Project Historian will have a public server for the occasional user.

6.1.1. Requirements

Project Historian has been tested to work on a virtual server with 2 CPU cores, 4GB of memory and 30GB of storage, which is easily satisfied by modern laptops. A persistent power and network connection is preferred, however, as Project Historian's offline part involves scheduled tasks that requires Internet access.

6.1.2. Set-up

Project Historian's set-up process involves four steps outlined below.

1. Download and extract the code in a user-chosen folder.

2. Install the required Python packages.
3. Schedule `cron` jobs to run the RSS caching, preprocessing and model training on a daily basis.
4. Run the server using a provided script. Project Historian can now be used from the web interface, although it will take some time for Project Historian to cache enough data to be useful.

The actual code and instructions for the above can be found in the attached repository.

6.2. Usage

This section will demonstrate the usage of Project Historian with a sample query about the events in the Russia investigation.

6.2.1. Invoking the interface

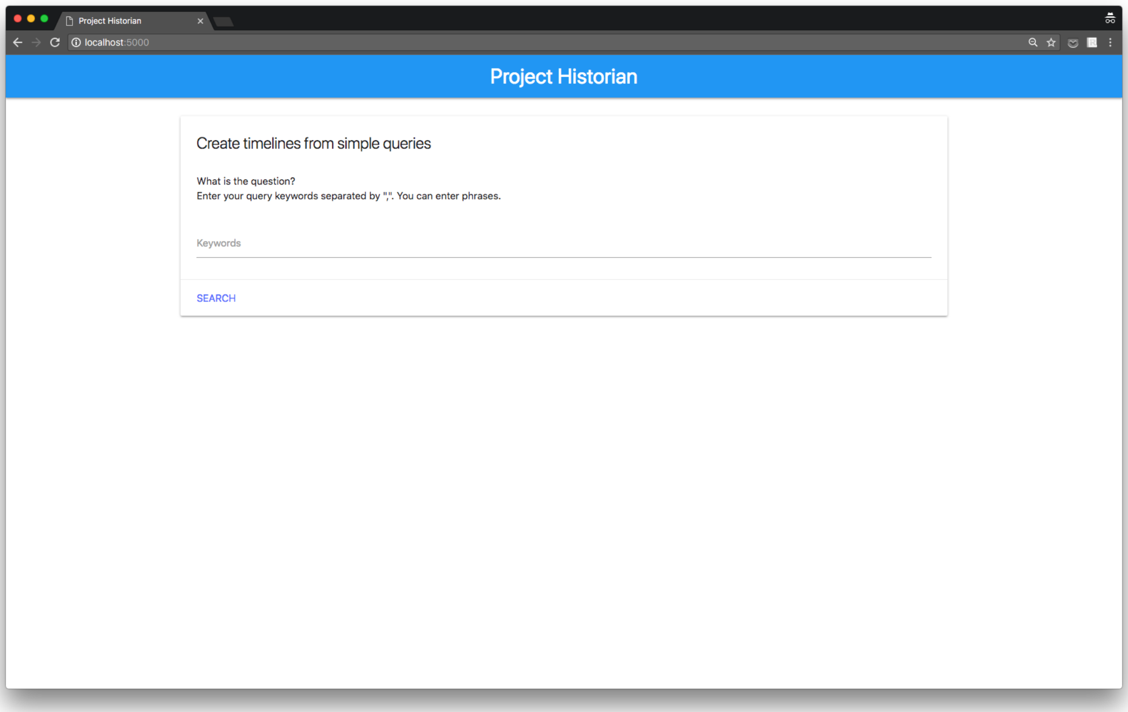
Project Historian runs on a web interface that is hosted on a server. To access a locally running server, the user can open their browser and go to http://localhost:5000, where they will be able to see Project HistorianâĂŹs user interface.

6.2.2. Making a query

The user makes a query by typing into the search box. In this case, the user can type "trump, russia", and then click the "Search" link below.

6.2.3. Examining the expanded keywords

Project Historian will return a list of keywords that it thinks will be useful in expanding the search. For example, it suggests keywords as shown in the image below.



Create timelines from simple queries

What is the question?
Enter your query keywords separated by ",". You can enter phrases.

Keywords

Figure 6.1.: Project Historian's web interface.

Each expanded keywords can be dismissed by clicking the "x" button next to it if the user finds it irrelevant. In this case, for example, the user can take out "donald" and "magoo", since the other keywords have captured the meaning well enough.

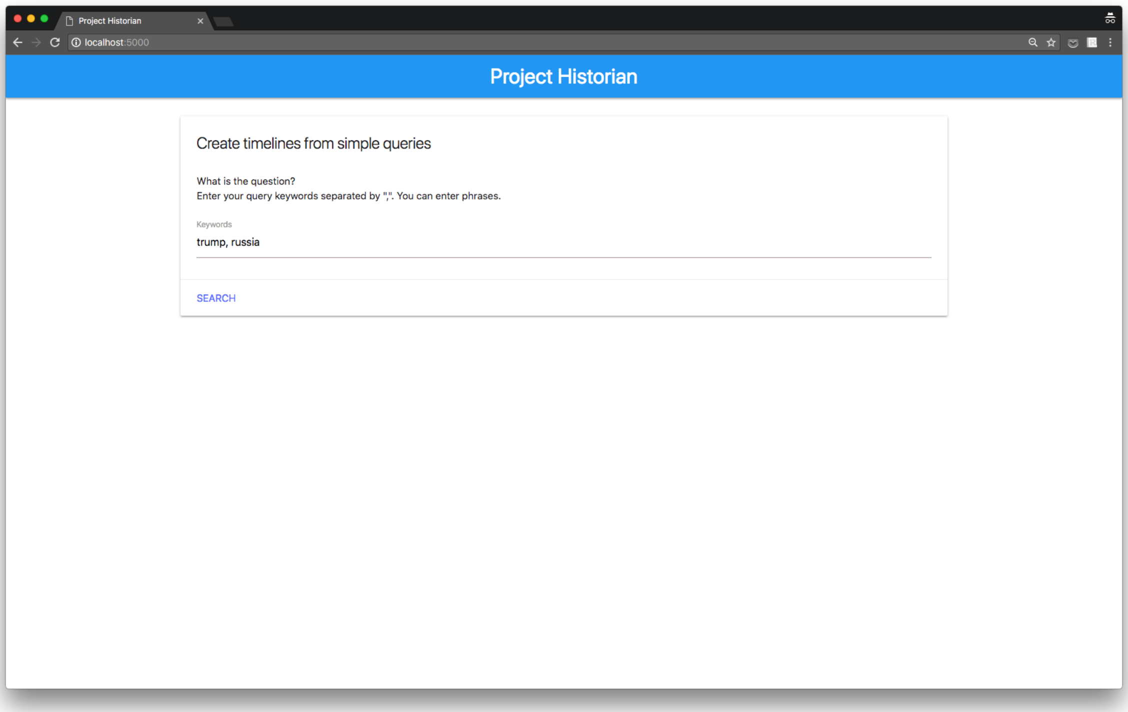
After cleaning up the expanded keywords, the user can click on "continue" to make the final query to Project Historian.

6.2.4. Viewing and editing the timeline data

After a few seconds to a few minutes of processing, during which a progress bar will be shown, Project Historian will show the initial timeline data.

Each event can be expanded to show the stories within by clicking on the headline.

The user will be able to select the events and stories that go into the final output with the checkboxes to the left of each entry. By default, all output is selected. The user will also be able to merge clusters by dragging one cluster on top of another, and rename clusters using an input field visible in each expanded cluster.



Create timelines from simple queries

What is the question?
Enter your query keywords separated by ",". You can enter phrases.

trump, russia

Figure 6.2.: Making a query in Project Historian.

6.2.5. Making the final timeline

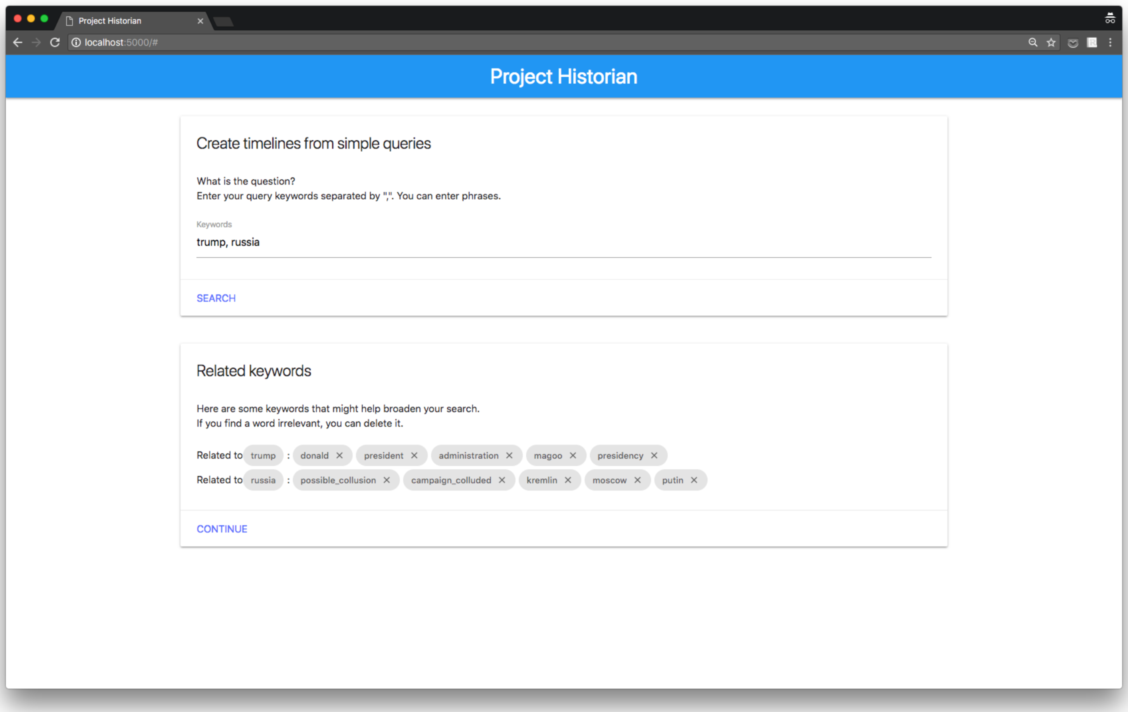
After the user finishes refining the timeline's output, she can click "make the timeline" at the bottom of the section to construct the final timeline, which is more visually straightforward than the initial timeline data.

Each section can be expanded to reveal a horizontal timeline of the stories within the section.

If the user decides to further refine the timeline output, she can go back to the initial results and continue editing there. Once she finishes, the "make the timeline" option will generate a new timeline with the latest edits.

6.2.6. Exporting the result

Once the final timeline is generated, the user has the option to export the graphics as well as the data.


Project Historian

localhost:5000/#

Project Historian

Create timelines from simple queries

What is the question?
Enter your query keywords separated by ",". You can enter phrases.

Keywords
trump, russia

SEARCH

Related keywords

Here are some keywords that might help broaden your search.
If you find a word irrelevant, you can delete it.

Related to trump : donald ✕ president ✕ administration ✕ magoo ✕ presidency ✕
Related to russia : possible_collusion ✕ campaign_colluded ✕ kremlin ✕ moscow ✕ putin ✕

CONTINUE

Figure 6.3.: Managing related keywords in Project Historian.

Exporting the graphics

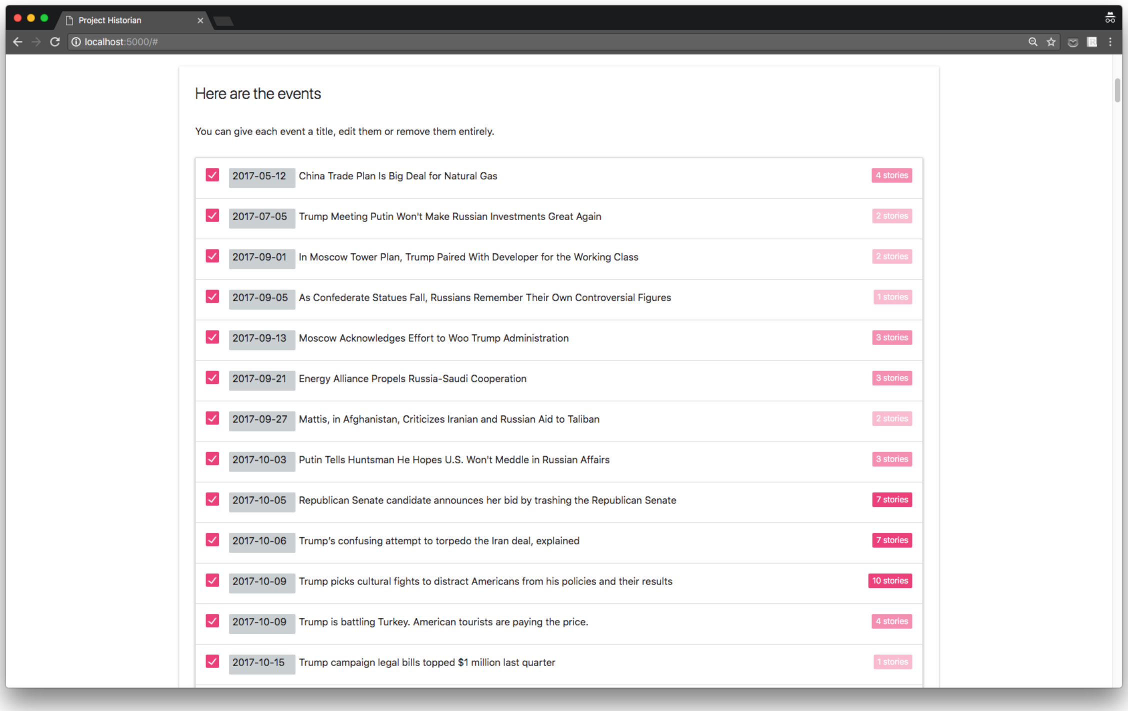
The user can export the interactive timeline into an HTML file, which can be opened directly to view the final timeline by itself. The code contained in the HTML file can also be embedded into an online news story as an interactive element.

Exporting the data in JSON format

The user also has the option to export data in JSON (JavaScript Object Notation) format, a machine-readable data format that facilitates easy custom visualization via Javascript. The data is downloaded as a text file that contains the encoded data.

Exporting the data as Excel

The user can also export the data as an Excel spreadsheet for a familiar reading experience. Each event cluster is organized as worksheet that contains the stories within.

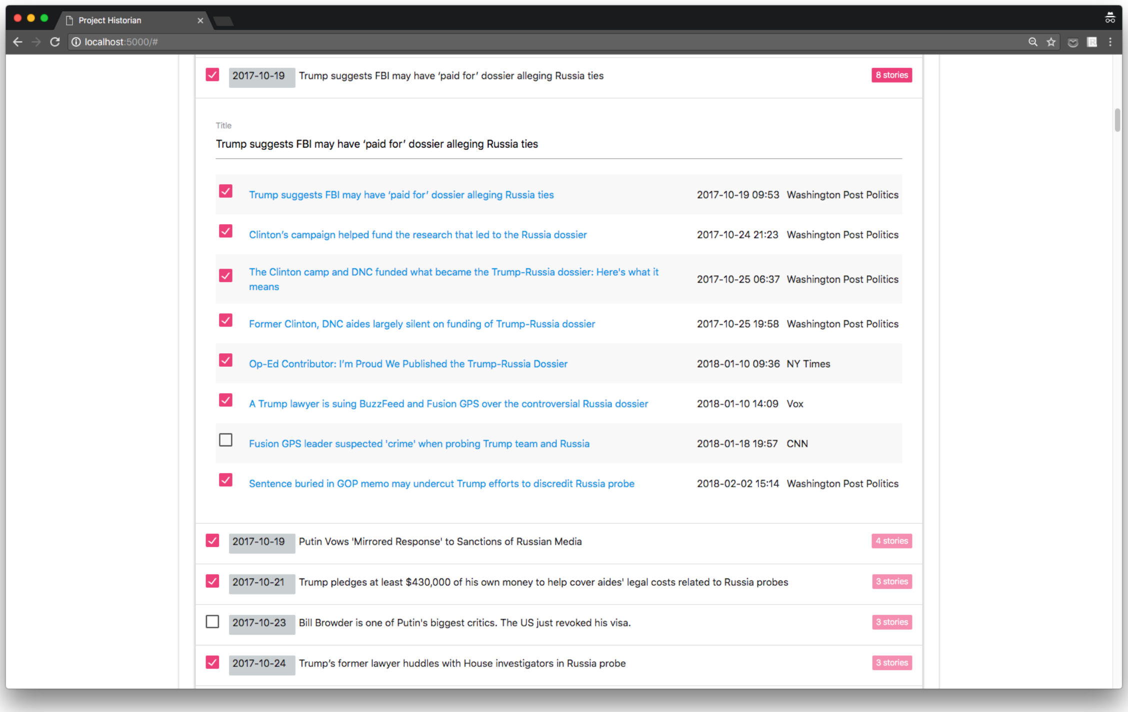

Project Historian
localhost:5000/#

Here are the events

You can give each event a title, edit them or remove them entirely.

✓	2017-05-12	China Trade Plan Is Big Deal for Natural Gas	4 stories
✓	2017-07-05	Trump Meeting Putin Won't Make Russian Investments Great Again	2 stories
✓	2017-09-01	In Moscow Tower Plan, Trump Paired With Developer for the Working Class	2 stories
✓	2017-09-05	As Confederate Statues Fall, Russians Remember Their Own Controversial Figures	1 stories
✓	2017-09-13	Moscow Acknowledges Effort to Woo Trump Administration	3 stories
✓	2017-09-21	Energy Alliance Propels Russia-Saudi Cooperation	3 stories
✓	2017-09-27	Mattis, in Afghanistan, Criticizes Iranian and Russian Aid to Taliban	2 stories
✓	2017-10-03	Putin Tells Huntsman He Hopes U.S. Won't Meddle in Russian Affairs	3 stories
✓	2017-10-05	Republican Senate candidate announces her bid by trashing the Republican Senate	7 stories
✓	2017-10-06	Trump's confusing attempt to torpedo the Iran deal, explained	7 stories
✓	2017-10-09	Trump picks cultural fights to distract Americans from his policies and their results	10 stories
✓	2017-10-09	Trump is battling Turkey. American tourists are paying the price.	4 stories
✓	2017-10-15	Trump campaign legal bills topped $1 million last quarter	1 stories

Figure 6.4.: Initial timeline data in Project Historian


Project Historian
localhost:5000/#

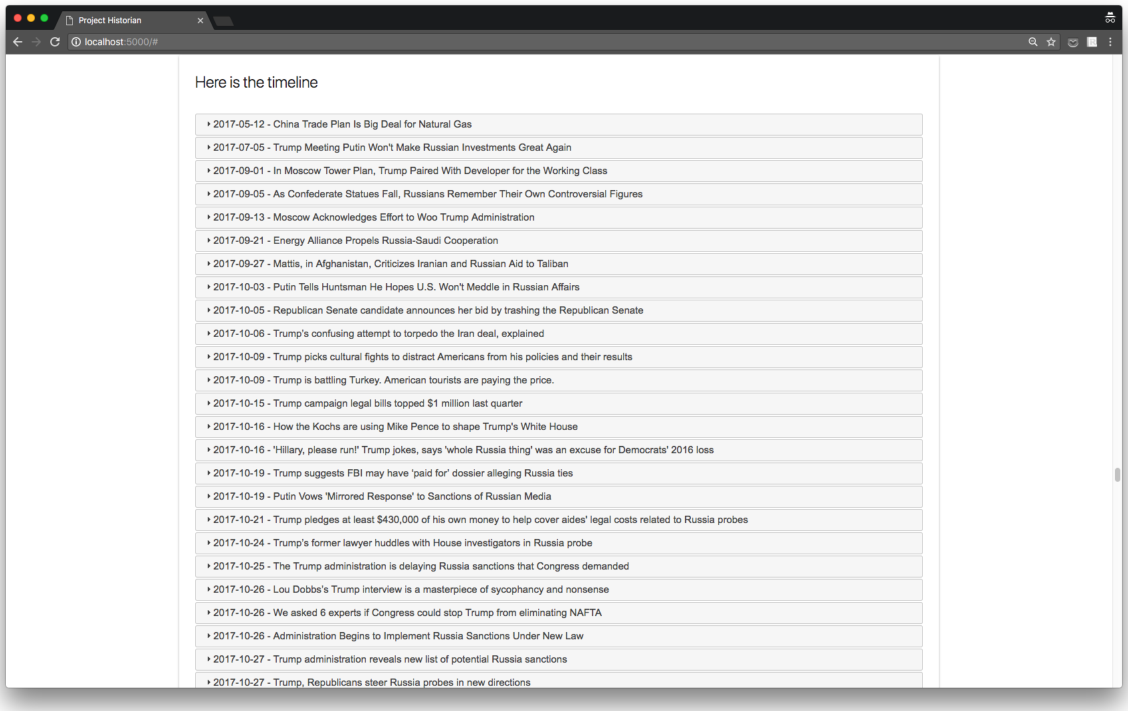
✓	2017-10-19	Trump suggests FBI may have 'paid for' dossier alleging Russia ties	8 stories

Title
Trump suggests FBI may have 'paid for' dossier alleging Russia ties

✓	Trump suggests FBI may have 'paid for' dossier alleging Russia ties	2017-10-19 09:53	Washington Post Politics
✓	Clinton's campaign helped fund the research that led to the Russia dossier	2017-10-24 21:23	Washington Post Politics
✓	The Clinton camp and DNC funded what became the Trump-Russia dossier: Here's what it means	2017-10-25 06:37	Washington Post Politics
✓	Former Clinton, DNC aides largely silent on funding of Trump-Russia dossier	2017-10-25 19:58	Washington Post Politics
✓	Op-Ed Contributor: I'm Proud We Published the Trump-Russia Dossier	2018-01-10 09:36	NY Times
✓	A Trump lawyer is suing BuzzFeed and Fusion GPS over the controversial Russia dossier	2018-01-10 14:09	Vox
☐	Fusion GPS leader suspected 'crime' when probing Trump team and Russia	2018-01-18 19:57	CNN
✓	Sentence buried in GOP memo may undercut Trump efforts to discredit Russia probe	2018-02-02 15:14	Washington Post Politics

✓	2017-10-19	Putin Vows 'Mirrored Response' to Sanctions of Russian Media	4 stories
✓	2017-10-21	Trump pledges at least $430,000 of his own money to help cover aides' legal costs related to Russia probes	3 stories
☐	2017-10-23	Bill Browder is one of Putin's biggest critics. The US just revoked his visa.	3 stories
✓	2017-10-24	Trump's former lawyer huddles with House investigators in Russia probe	3 stories

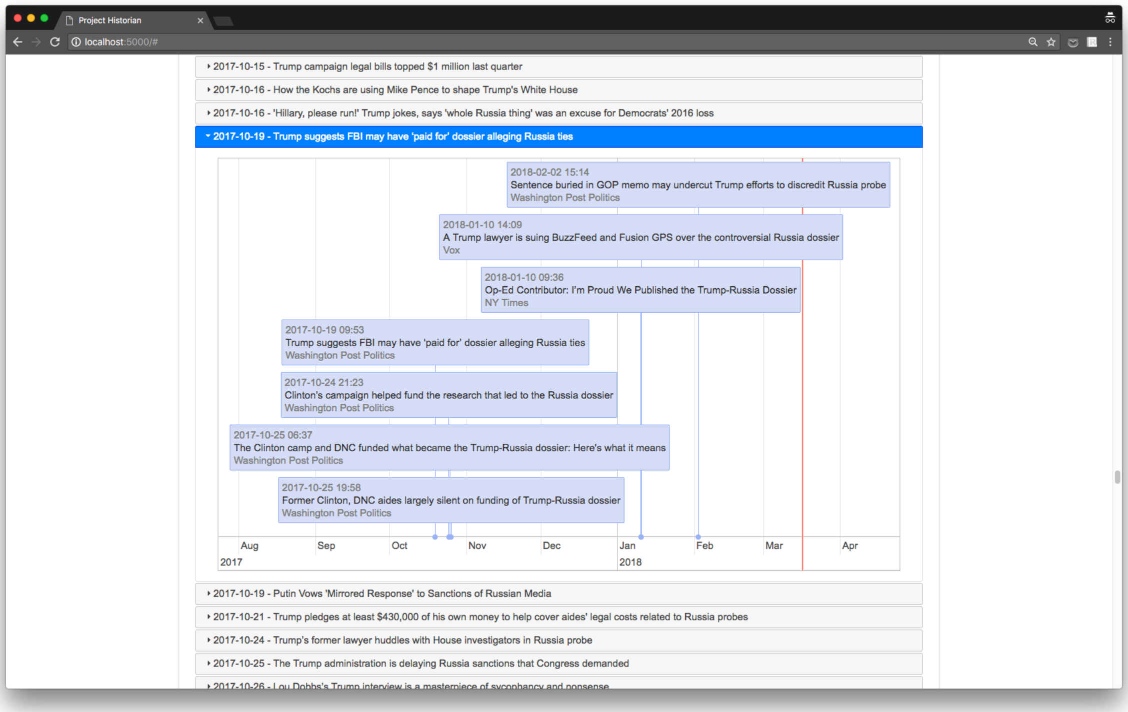
Figure 6.5.: Editing timeline data in Project Historian.



Here is the timeline

- 2017-05-12 - China Trade Plan Is Big Deal for Natural Gas
- 2017-07-05 - Trump Meeting Putin Won't Make Russian Investments Great Again
- 2017-09-01 - In Moscow Tower Plan, Trump Paired With Developer for the Working Class
- 2017-09-05 - As Confederate Statues Fall, Russians Remember Their Own Controversial Figures
- 2017-09-13 - Moscow Acknowledges Effort to Woo Trump Administration
- 2017-09-21 - Energy Alliance Propels Russia-Saudi Cooperation
- 2017-09-27 - Mattis, in Afghanistan, Criticizes Iranian and Russian Aid to Taliban
- 2017-10-03 - Putin Tells Huntsman He Hopes U.S. Won't Meddle in Russian Affairs
- 2017-10-05 - Republican Senate candidate announces her bid by trashing the Republican Senate
- 2017-10-06 - Trump's confusing attempt to torpedo the Iran deal, explained
- 2017-10-09 - Trump picks cultural fights to distract Americans from his policies and their results
- 2017-10-09 - Trump is battling Turkey. American tourists are paying the price.
- 2017-10-15 - Trump campaign legal bills topped $1 million last quarter
- 2017-10-16 - How the Kochs are using Mike Pence to shape Trump's White House
- 2017-10-16 - 'Hillary, please run!' Trump jokes, says 'whole Russia thing' was an excuse for Democrats' 2016 loss
- 2017-10-19 - Trump suggests FBI may have 'paid for' dossier alleging Russia ties
- 2017-10-19 - Putin Vows 'Mirrored Response' to Sanctions of Russian Media
- 2017-10-21 - Trump pledges at least $430,000 of his own money to help cover aides' legal costs related to Russia probes
- 2017-10-24 - Trump's former lawyer huddles with House investigators in Russia probe
- 2017-10-25 - The Trump administration is delaying Russia sanctions that Congress demanded
- 2017-10-26 - Lou Dobbs's Trump interview is a masterpiece of sycophancy and nonsense
- 2017-10-26 - We asked 6 experts if Congress could stop Trump from eliminating NAFTA
- 2017-10-26 - Administration Begins to Implement Russia Sanctions Under New Law
- 2017-10-27 - Trump administration reveals new list of potential Russia sanctions
- 2017-10-27 - Trump, Republicans steer Russia probes in new directions

Figure 6.6.: The output timeline in Project Historian.



- 2017-10-15 - Trump campaign legal bills topped $1 million last quarter
- 2017-10-16 - How the Kochs are using Mike Pence to shape Trump's White House
- 2017-10-16 - 'Hillary, please run!' Trump jokes, says 'whole Russia thing' was an excuse for Democrats' 2016 loss
- 2017-10-19 - Trump suggests FBI may have 'paid for' dossier alleging Russia ties

2018-02-02 15:14
Sentence buried in GOP memo may undercut Trump efforts to discredit Russia probe
Washington Post Politics

2018-01-10 14:09
A Trump lawyer is suing BuzzFeed and Fusion GPS over the controversial Russia dossier
Vox

2018-01-10 09:36
Op-Ed Contributor: I'm Proud We Published the Trump-Russia Dossier
NY Times

2017-10-19 09:53
Trump suggests FBI may have 'paid for' dossier alleging Russia ties
Washington Post Politics

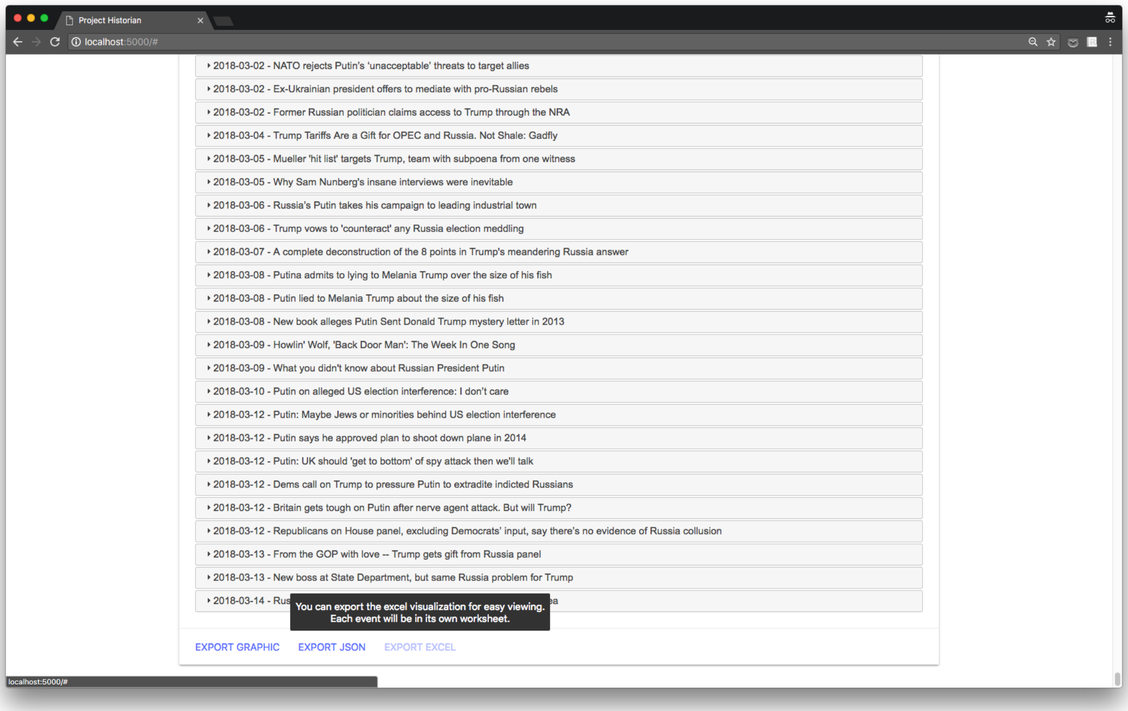
2017-10-24 21:23
Clinton's campaign helped fund the research that led to the Russia dossier
Washington Post Politics

2017-10-25 06:37
The Clinton camp and DNC funded what became the Trump-Russia dossier: Here's what it means
Washington Post Politics

2017-10-25 19:58
Former Clinton, DNC aides largely silent on funding of Trump-Russia dossier
Washington Post Politics

- 2017-10-19 - Putin Vows 'Mirrored Response' to Sanctions of Russian Media
- 2017-10-21 - Trump pledges at least $430,000 of his own money to help cover aides' legal costs related to Russia probes
- 2017-10-24 - Trump's former lawyer huddles with House investigators in Russia probe
- 2017-10-25 - The Trump administration is delaying Russia sanctions that Congress demanded
- 2017-10-26 - Lou Dobbs's Trump interview is a masterpiece of sycophancy and nonsense

Figure 6.7.: An expanded event timeline in Project Historian.



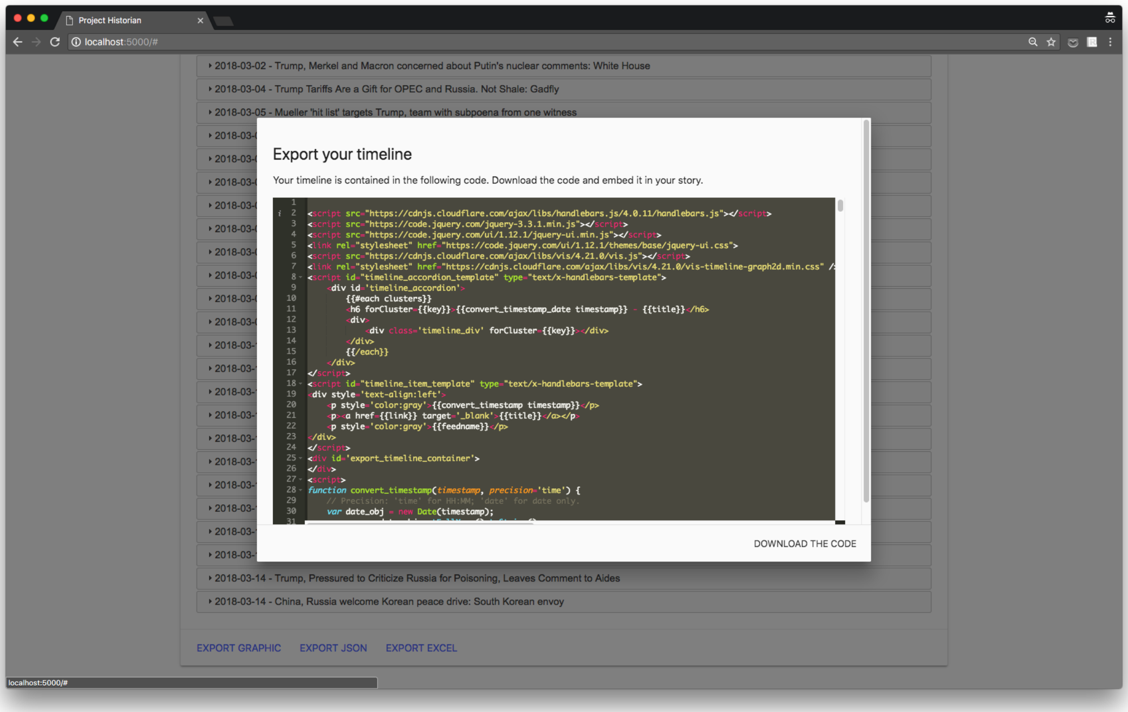
Project Historian

localhost:5000/#

- 2018-03-02 - NATO rejects Putin's 'unacceptable' threats to target allies
- 2018-03-02 - Ex-Ukrainian president offers to mediate with pro-Russian rebels
- 2018-03-02 - Former Russian politician claims access to Trump through the NRA
- 2018-03-04 - Trump Tariffs Are a Gift for OPEC and Russia. Not Shale: Gadfly
- 2018-03-05 - Mueller 'hit list' targets Trump, team with subpoena from one witness
- 2018-03-05 - Why Sam Nunberg's insane interviews were inevitable
- 2018-03-06 - Russia's Putin takes his campaign to leading industrial town
- 2018-03-06 - Trump vows to 'counteract' any Russia election meddling
- 2018-03-07 - A complete deconstruction of the 8 points in Trump's meandering Russia answer
- 2018-03-08 - Putina admits to lying to Melania Trump over the size of his fish
- 2018-03-08 - Putin lied to Melania Trump about the size of his fish
- 2018-03-08 - New book alleges Putin Sent Donald Trump mystery letter in 2013
- 2018-03-09 - Howlin' Wolf, 'Back Door Man': The Week In One Song
- 2018-03-09 - What you didn't know about Russian President Putin
- 2018-03-10 - Putin on alleged US election interference: I don't care
- 2018-03-12 - Putin: Maybe Jews or minorities behind US election interference
- 2018-03-12 - Putin says he approved plan to shoot down plane in 2014
- 2018-03-12 - Putin: UK should 'get to bottom' of spy attack then we'll talk
- 2018-03-12 - Dems call on Trump to pressure Putin to extradite indicted Russians
- 2018-03-12 - Britain gets tough on Putin after nerve agent attack. But will Trump?
- 2018-03-12 - Republicans on House panel, excluding Democrats' input, say there's no evidence of Russia collusion
- 2018-03-13 - From the GOP with love -- Trump gets gift from Russia panel
- 2018-03-13 - New boss at State Department, but same Russia problem for Trump
- 2018-03-14 - Rus

You can export the excel visualization for easy viewing.
Each event will be in its own worksheet.

Figure 6.8.: Export options in Project Historian.



Project Historian

localhost:5000/#

- 2018-03-02 - Trump, Merkel and Macron concerned about Putin's nuclear comments: White House
- 2018-03-04 - Trump Tariffs Are a Gift for OPEC and Russia. Not Shale: Gadfly
- 2018-03-05 - Mueller 'hit list' targets Trump, team with subpoena from one witness
- 2018-03-14 - Trump, Pressured to Criticize Russia for Poisoning, Leaves Comment to Aides
- 2018-03-14 - China, Russia welcome Korean peace drive: South Korean envoy

Export your timeline

Your timeline is contained in the following code. Download the code and embed it in your story.

```

<script src="https://cdnjs.cloudflare.com/ajax/libs/handlebars.js/4.0.11/handlebars.js"></script>
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<script src="https://code.jquery.com/ui/1.12.1/jquery-ui.min.js"></script>
<link rel="stylesheet" href="https://code.jquery.com/ui/1.12.1/themes/base/jquery-ui.css">
<script src="https://cdnjs.cloudflare.com/ajax/libs/vis/4.21.0/vis.js"></script>
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/vis/4.21.0/vis-timeline-graph2d.min.css" /
<script id="timeline_accordion_template" type="text/x-handlebars-template">
    <div id='timeline_accordion'>
        {{#each clusters}}
        <h6 forCluster={{key}}>{{convert_timestamp_date timestamp}} - {{title}}</h6>
        <div>
            <div class='timeline_div' forCluster={{key}}></div>
        </div>
        {{/each}}
    </div>
</script>
<script id="timeline_item_template" type="text/x-handlebars-template">
<div style='text-align:left'>
    <p style='color:gray'>{{convert_timestamp timestamp}}</p>
    <p><a href={{link}} target='_blank'>{{title}}</a></p>
    <p style='color:gray'>{{feedname}}</p>
</div>
</script>
<div id='export_timeline_container'>
</div>
<script>
function convert_timestamp(timestamp, precision='time') {
    // Precision: 'time' for HH:MM; 'date' for date only.
    var date_obj = new Date(timestamp);
```

Figure 6.9.: Exporting the timeline as interactive graphics.

Chapter 7.

Conclusion

Project Historian presents a unique solution to a common research and context problem in journalism today. By breaking down news coverage into event timelines, it makes a constant stream of news much more digestable for journalists, readers and scholars alike. The project's extensive use of unsupervised learning techniques means that it is easily adapted to any source of news content that can provide news story text, making it easy for interested publications to integrate. It also ensures that any data it is based on is held to the journalistic standards of the data's sources. In its current implementation, which is based on RSS feeds of reputable publications, it has already achieved promising performance. This performance, will only improve with richer data sources and deeper archives.

The most significant aspect of Project Historian is the overall pipeline for timeline retrieval, rather than the specific technology used. In order to achieve acceptable responsiveness with limited resource during its development, Project Historian currently chooses speed over sophistication for each stage of it natural language processing. This trade-off covers problems such as phrase detection, word representation model training, salient information extraction and the clustering algorithm. Future research to improve Project Historian can concentrate on optimizing each of these stages to improve the performance of the overall package, and therefore make it a more valuable tool for journalists, publications and scholars.

Appendix A.

Technical Appendix

A.1. Database format for Project Historian

Project Historian's main database uses the table format as shown in Table A.1 to store RSS information.

The columns in use are `title`, `content`, `summary`, `feedname`, `canonical`, `published` and `preprocessed`. `preprocessed` is derived from `title`, `content` and `summary`, which come from the RSS data.

If a news organization hopes to integrate Project Historian internally, it can replace the RSS database with a database based on its archive and populate the columns in use mentioned above except for `preprocessed`, which is generated by Project Historian.

A.2. Clustering algorithm

Project Historian uses the following clustering algorithm to perform event clustering.

Column name	Data Type	Description
`id`	`TEXT`	A unique indentifier for each entry
`title`	`TEXT`	Title of each feed entry
`feedname`	`TEXT`	The display name for the RSS feed (e.g. WSJ Business)
`content`	`TEXT`	The content of the feed entry, usually an article snippet
`summary`	`TEXT`	The summary of the feed entry
`author`	`TEXT`	The author name for the feed entry
`published`	`INTEGER`	A UNIX timestamp of the feed entry's publication time
`visual`	`TEXT`	When present, the visual assets (e.g. thumbnails) associated with the feed entry in JSON format
`engagement`	`INTEGER`	An indicator of the popularity of the entry from the Feedly Streams API
`canonical`	`TEXT`	Link to the feed entry's corresponding article
`alternate`	`TEXT`	Alternate links to the feed entry's corresponding article in JSON format
`enclosure`	`TEXT`	A collection of media object links (e.g. video, audio) of the feed entry in JSON format
`cachedate`	`TEXT`	The latest date on which the item is cached
`preprocessed`	`TEXT`	Preprocessed text (after tokenization and phrase detection steps) including the title, summary and content.

Table A.1.: The database format for Project Historian.

```
Data: A list of vectors $\{v_i\}$, each representing a story
Result: A list of clusters $\{C_i\}$, with each cluster including a list of indices in $\{v_i\}$.
let $Q$ be an empty queue;
let $C$ be an empty set;
enqueue $\{1, 2, \ldots, n\}$ in $Q$;
while $Q$ is not empty do
    dequeue $C_t$ from $Q$;
    if $C_t$ is a high-quality cluster then
        add $C_t$ to $C$;
        continue;
    else
        use K-Means to split $C_t$ into two clusters, $C_{t1}$ and $C_{t2}$;
        enqueue $C_{t1}$ in $Q$;
        enqueue $C_{t2}$ in $Q$;
    end
end
return $C$
```

Algorithm 1: The clustering algorithm in Project Historian

Bibliography

[1] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 133–142, 2003.

[2] Danqi Chen and Christopher Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 740–750, 2014.

[3] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 363–370. Association for Computational Linguistics, 2005.

[4] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[5] David M Blei. Probabilistic topic models. *Communications of the ACM*, 55(4):77–84, 2012.

[6] Yoav Goldberg. Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1):1–309, 2017.

[7] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *hlt-Naacl*, volume 13, pages 746–751, 2013.

[8] Brandon Roberts. Extracting references from political speech auto-transcripts. Paper available at `https://sites.google.com/view/dsandj2017/accepted-papers?authuser=0`, 2017.

[9] Northwerstern University Knight Lab. Timeline. `http://timeline.knightlab.com`.

[10] Will Ripley. Tearful north korean waitresses: Our 'defector' colleagues were tricked. http://www.cnn.com/2016/04/20/asia/north-korea-restaurant-defectors/index.html.

[11] TIME staff. Nelson mandelaâĂŹs extraordinary life: An interactive timeline. http://world.time.com/2013/12/05/nelson-mandelas-extraordinary-life-an-interactive-timeline/.

[12] Daniel Cooper. The collapse of microsoft and nokia's mobile business. https://www.engadget.com/2016/04/22/microsoft-mobile-timeline/.

[13] Eugene Mustfinn. Time graphics. https://time.graphics.

[14] Vizzlo GmbH. Vizzlo. https://vizzlo.com.

[15] Mnemograph LLC. Timeglider: web-based timeline software. http://timeglider.com.

[16] Almende B.V. vis.js - a dynamic, browser based visualization library. http://visjs.org.

[17] Johnny Harris. How does a 7-minute video about syria get over 100m views?, 2017. https://storytelling.voxmedia.com/2017/4/18/15325756/vox-syria-video-behind-the-scenes.

[18] Richard Hipp. Sqlite. https://sqlite.org.

[19] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, 2009.

[20] Gillinder Bedi, Facundo Carrillo, Guillermo A Cecchi, Diego Fernández Slezak, Mariano Sigman, Natália B Mota, Sidarta Ribeiro, Daniel C Javitt, Mauro Copelli, and Cheryl M Corcoran. Automated analysis of free speech predicts psychosis onset in high-risk youths. *npj Schizophrenia*, 1:15030, 2015.

[21] Daniel Pletea, Bogdan Vasilescu, and Alexander Serebrenik. Security and emotion: sentiment analysis of security discussions on github. In *Proceedings of the 11th working conference on mining software repositories*, pages 348–351. ACM, 2014.

[22] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. http://is.

muni.cz/publication/884893/en.

[23] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[24] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[25] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[27] Armin Ronacher, Adam Zapletal, Ali Afshar, Chris Edgemon, Chris Grindstaff, Christopher Grebs, Daniel NeuhÃd'user, Dan Sully, David Lord, Edmond Burnett, Florent Xicluna, Georg Brandl, Jeff Widman, Justin Quick, Kenneth Reitz, Keyan Pishdadian, Marian Sigler, Martijn Pieters, Matt Campell, Matthew Frazier, Michael van Tellingen, Ron DuPlain, Sebastien Estienne, Simon Sapin, Stephane Wirtel, Thomas Schranz, and Zhao Xiaohong. Flask. http://flask.pocoo.org/docs/0.12/.

[28] Alvin Wang, Alan Chang, Alex Mark, and Kevin Louie. Materialize. http://next.materializecss.com/.

[29] Amit Singhal et al. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24(4):35–43, 2001.

List of figures

List of tables