DIGITAL ACCESS TO SCHOLARSHIP AT HARVARD

DASH.HARVARD.EDU

Cross-Language News Article Clustering

Link

http://nrs.harvard.edu/urn-3:HUL.InstRepos:37736777

Terms of use

This article was downloaded from Harvard University's DASH repository, and is made available under the terms and conditions applicable to Other Posted Material (LAA), as set forth at

https://harvardwiki.atlassian.net/wiki/external/NGY5NDE4ZjgzNTc5NDQzMGIzZWZhMGFlOWI2M2EwYTg

Accessibility

https://accessibility.huit.harvard.edu/digital-accessibility-policy

Share Your Story

The Harvard community has made this article openly available.
Please share how this access benefits you. Submit a story

Cross-Language News Article Clustering

Nathan S. Guerin

A Thesis in the Field of Software Engineering

for the Degree of Master of Liberal Arts in Extension Studies

Harvard University

November, 2017

© 2017 Nathan S. Guerin

Abstract

This thesis describes a method of delivering topically-clustered English and Chinese news articles for monolingual readers and provides a fully-implemented application. In today's highly-polarized political climate, we are inundated with a diversity of opinions in television and online news media markets. Yet there are some topics, particularly those pertaining to foreign policy, in which a nation's news media exhibits bias by nature of who's reporting the news and to whom it's being reported. One potential way for the media's audience to counteract bias is by comparing and contrasting news articles about the same topic written in different languages and different countries. Such comparisons can expose unique perspectives by nature of their origin.

The application developed for this thesis allows one to quickly identify articles about the same topic in different languages. It does this by clustering news articles by topic and presenting them in groups. For monolingual readers, the application integrates with Google Translate to provide a translated version of the source text. In order to provide these services, the application scrapes Chinese and English news articles from the web, extracts their relevant features, translates these features into a common human language, uses machine-learning techniques to reduce the dimensionality of the features, and stores those features for on-demand clustering and similar article retrieval.

This thesis and similar projects have many possible applications, from providing the casual bilingual reader the chance to explore news coverage from different viewpoints, to use by researchers in both the US and China in better understanding the media and how it shapes public opinion. Both the application and its relevant source code are accessible on the author's website.

Author's Biographical Sketch

Nathan Guerin is a software engineer currently living and working in Raleigh, North Carolina. His professional experiences include designing and developing software systems in the healthcare and travel industries. He greatly enjoys teaching and has been a Teaching Assistant for multiple courses taught through Harvard's Division of Continuing Education.

Nathan has lived and worked on four different continents and is very passionate about language and culture. Of the foreign languages he speaks, he finds Chinese the most interesting and is an avid hobbyist Chinese-to-English translator, some of the products of which can be seen on his website.[1] His passion for exploring how software can be used to lower linguistic and cultural barriers was the driving impetus for his choosing the topic of the thesis you're currently reading.

[1] `http://www.plaintexttransmissions.org/`

Dedication

To Emily, for her steadfast support of my educational pursuits over the past few years and for not leaving even though we gave up far too many evening and weekends together,

to Rich and Connie, for teaching that anything's possible and encouraging me to travel to far-off places,

to Missy, for being a great sister and bringing Matt and Mylo into our lives,

and to Josie, Mark, Schuyler and Lucas, for helping Emily and me immeasurably over the past year in Cambridge.

Acknowledgments

I am deeply indebted to my thesis adviser, Dr. James Frankel. The bi-weekly meetings we held over the past year kept me and this thesis on track. His advice and questions were invaluable and often pointed me towards new, fruitful paths of inquiry and development. The first of the three courses I took of his, Operating Systems, was the course that inspired me to formally enroll in HES and complete the Software Engineering masters degree. I'll never forget the great feeling he left us with in his classes upon first really understanding how different parts of computing systems work.

Dr. Jeff Parker helped me immeasurably during the early stages of this thesis. The phase of brainstorming thesis topics led to a few different ideas I was interested in exploring. Dr. Parker gave me a lot of great suggestions, connected me with different professors and students who were experts in the fields I was exploring, and nudged me away from some of my weaker ideas. The Bioinformatics Algorithms course Dr. Parker taught was a great first introduction to applications in research computing, an area I've since become very interested in. I am also thankful for the multiple enrollment throughout my time at HES that Dr. Parker helped me administratively, academically, and professionally.

My friend Dr. Giorgio Strafella of St. Gallen University was also very helpful during the thesis writing process. Dr. Strafella is an expert in modern Chinese literature and society. After deciding that I would like to work on a topic involving modern Chinese society, Dr. Strafella and I exchanged many messages brainstorming concrete topic ideas. After I settled on a topic, Dr. Strafella was always available whenever I had questions pertaining to contemporary Chinese society.

Contents

List of Tables

List of Figures

Chapter 1

Why Is This Application Necessary?

While it's impossible to predict the future, the general consensus of the political and foreign policy establishment is that the US-Chinese bilateral relationship is going to be the most important international relationship in the 21st century. In the words of China's current premier Xi Jinping, when "China and the US have a good relationship it's beneficial not only to the people of these two nations, but also to the world. Cooperation is the only choice for China and the US" (Foreign Ministry of the People's Republic of China [FMPRC], 2017). American president Donald Trump has echoed this idea: "The responsibilities of the US and China are great, both sides should continue to communicate and cooperate on important issues, and by doing so will be able to accomplish great things together" (FMPRC, 2017). Yet there will be many challenges, the chief among them mutual mistrust, that have the potential to make managing this relationship difficult.

Competitive antagonism impedes the ability of the US and China to cooperate. Simply put, China is becoming a superpower, rising into the category of global political power that the US has been the sole occupant of since the disintegration of the Soviet Union. The director of Harvard University's Belfer Center for Science and International Affairs, Graham Allison has identified 16 cases over the past 500 years where two country's relative power dynamic has closely resembled that of the US and China, and in

12 of those 16 cases the outcome has been war (Pazzanese, 2017). Issues such as trade grievances, claims of containment, tangled and complex alliances, and fundamentally divergent visions for the coming century all encourage antagonism. Fingar and Fan (2013) warn that "complacency and failure to address misperceptions and mistrust […] will have unfortunate consequences for both sides" (p. 125). Both the US and China are highly nationalistic countries which can make it difficult to compromise on certain issues. It's difficult to imagine the US and China compromising on Taiwan or human rights, for example. While it is inaccurate and often a vast over-generalization to ascribe qualities to all the people of a country, the different educational, ideological, religious, and linguistic systems can make people-to-people mutual understanding a challenge. Technology that assists people from both countries to better understand the other's perspective is one positive step towards dispelling misperceptions and building trust. The main goal of this thesis is to provide a tool that people of both countries can use to better understand each other.

Yet, we easily fall prey to selective exposure—the process of filtering out messages that do no match our beliefs because of political preference, or simply by selecting a certain information medium. It's a challenge to transcend the perspectives we are most familiar with, even more so when they are cloaked behind a language barrier. Some of the underlying reasons for this are covered in Munson, Lee, and Resnick (2013), wherein they create a browser extension encouraging people to look at different perspectives. In their article, they point out that selective exposure is simply easier and avoids cognitive dissonance (p. 1). They also warn that "exposure to diverse views is a necessary ingredient in deliberative debate, which political theorists argue is necessary for a healthy democracy" (p. 2).

I would argue that selective exposure leads to negative outcomes in the more general case where two different parties, political or national, need to cooperate for mutual benefit. While the political system in the US has constitutional protections allowing the

free flow of information, many people have created self-censoring environments by falling into the pitfalls of selective exposure, especially when reading about topics such as international relations or trade policy. This is due to the fact that, as different as one argument may be from another, most citizens of a country reading international news are going to be reading it in the context of how it affects their country. It's just like reading an article in your local newspaper about last night's performance of the hometown sports team—coverage is going to be biased towards the home team. This is obvious and its reasons are self-explanatory. Fingar and Fan (2013) convincingly argue that reducing misperceptions requires frequently asking "what are you doing and why are you doing it" type of questions (p. 132). Short of speaking directly with someone in a foreign country, reading the news articles from that country's press can begin to answer these questions.

To facilitate the retrieval of news articles about the same topic in both English and Chinese so that, through the process of reading both sets of articles, a reader may possibly uncover previously unconsidered perspectives, this thesis develops an application that topically clusters English and Chinese news articles. The application has a web interface that is localized to both American English and Simplified Chinese. It crawls the web for articles and clusters them daily so that both Chinese or English readers can visit the website and browse that day's news. As the purpose of the application is to compare and contrast viewpoints, only those clusters that contain articles in both English and Chinese are presented to the user.

Chapter 2

Prior Work

This thesis is most similar in nature to two strains of work, one developed at Columbia University and another from a 2013 thesis by two master students of the Chalmers University of Technology in Gothenberg, Sweden.

Kathleen McKeown and her team at Columbia University developed an Information Retrieval/Natural Language Processing (IR/NLP) application entitled "Newsblaster" (Evans, Klavans, and McKeown, 2004; K. R. McKeown et al., 2002; K. McKeown et al., 2003; K. McKeown, Passonneau, Elson, Nenkova, and Hirschberg, 2005). Newsblaster crawled the web for news articles, grouped news articles together by topic, created text summaries of the resulting clusters, tracked stories across temporal and linguistic boundaries, evaluated the efficacy of machine-generated summaries versus human-generated summaries, and even expanded the article clustering to multiple languages. Their stated goal of the multilingual Newsblaster system is to allow users to compare articles about the same topic from multiple different perspectives that may arise from articles written in different languages—a goal quite similar to that of this thesis. While the Newsblaster website is still accessible on the internet as of August 2017, its final article scraping run was in January 2016 and did not include any non-English articles.

Conceptually, Newsblaster and this thesis share a common approach and goal. The

first four steps of Newsblaster's processing pipeline are the same as those followed in this thesis: (a) crawl news articles, (b) extract text, (c) translate, and (d) cluster. Multilingual Newsblaster then summarizes the clusters and classifies the summaries into common, predetermined news categories, such as world, sports, finance, etc. This thesis neither summarizes the articles nor attempts to classify the articles into predetermined categories. Unlike Newsblaster, this thesis places multilingual clustering front-and-center, whereas in Newsblaster it seemed to be a peripheral interest. It also localizes the interface and provides translations of articles to allow use by both English- and Chinese-speaking audiences.

David Evans was one of the doctoral students working on Newsblaster and its influence on his PhD dissertation (Evans, 2005) is clear. This work, "Identifying Similarity in Text: Multi-Lingual Analysis for Summarization," was primarily concerned with multi-lingual text similarity. In it, Evans created a framework called SimFinderML that allowed him to extract sentences of text in two different languages that are about the same thing. He focused on Arabic and English, but the framework he created can be extended to other languages and allows for manipulation of the different parameters used to calculate text similarity.

There are various applications for finding similar sentences in two documents about the same topic. Evans demonstrated two: better machine translation and the identification of sentences present in both articles and those that are unique to one article. The former application is based on the hypothesis that if two sentences are very similar, it's plausible that one can be replaced by the other. The latter application provides a starting point for exploring which parts of an article are unique to a language or culture and which are common across languages. Sentences present in one article but not in another may be indicative of divergent perspectives on an event.

This thesis borrowed many of the ideas from Evans' dissertation and applies them to cross-lingual article clustering. One of the more notable inspirations is the extraction of

different types of features, such as nouns, verbs, named entities, etc. from an article and the use of an empirical test to determine which features most accurately predict similar text. This thesis, like Evans' dissertation, is interested in building tools that facilitate the discovery of different perspectives across languages. The main differences between the two are that SimFinderML's scope in finding similarity is at the sentence level while this thesis' scope is at the article level, and that this thesis makes no attempt to offer a solution for better sentence translations.

A few years later in Sweden, Lönnberg and Yregård (2013) wrote a master's thesis on monolingual news article clustering. Their work focused mostly on algorithmic efficiency and online learning. They reviewed all of the major clustering algorithms, similarity measures, and preprocessing techniques—a helpful overview of the tools in the toolbox available for working on similar applications. They also developed their own clustering algorithm, called incremental clustering, and stated that it worked well, especially for the case where news articles are incrementally added to a dataset as they are published (p. 41). To measure the effectiveness of the incremental clustering algorithm, they classified a set of news articles and measured how well the various tested algorithms met their classification expectations. This thesis is similar to Lönnberg and Yregård's but one obvious difference is the inclusion of multiple languages. Additionally, Lönnberg and Yregård's thesis is concerned more with algorithmic analysis and efficiency while this thesis is more concerned about producing good clusters and developing an application to uncover and present interesting perspectives hidden behind different languages.

The next set of works similar to this thesis take computational approaches to designing information retrieval and presentation systems and attempt to counteract confirmation bias and selective exposure by encouraging readers to encounter diverse perspectives in the news. Two notable implementations are NewsCube by Park, Kang, Chung, and Song (2009) and the Balancer Chrome browser extension by Munson et al. (2013).

In their paper about NewsCube, Park et al. stated that their motivation for developing NewsCube was to mitigate the effects of media bias. They proposed achieving this by providing a user with multiple articles on the same topic from a variety of different perspectives. They defined the process of different news sources framing a story as the particular "aspects" of the story—one news source may emphasize one aspect of a story while another news source might emphasize another. NewsCube grouped different aspects together to create "aspect-based browsing." Park et al. also defined an extraction technique they called "News Structure-based Extraction," which applies discrete gradations of importance to different parts of a news article. For example, they noted that most articles adhere to the inverted pyramid model, so when grouping stories together News Structure-based Extraction increases the importance of the terms in the first paragraph. They found that Hierarchical Agglomerative Clustering (HAC) algorithms perform well in the unsupervised classification of news articles. To determine the effectiveness of their system, they had multiple user-groups evaluate the clustering results.

Like NewsCube, Munson et al. (2013) built a Chrome browser extension in an attempt to mitigate media bias and selective exposure. The browser extension "Balancer" provides users with a check on the bias of the information they're viewing. It classifies all English news sources on a scale from conservative to liberal and tracks browser usage. It then constructs weekly histograms of the political segments that each read news article falls into and, in certain cases, even suggests alternate sources to help balance news consumption. After conducting user trials, Munson et al. found that a minority of users viewed a small number of articles that fell outside of their normal conservative-liberal segment in the histogram. They also found that being exposed to the histogram inspired some users to explore more centrist news sources.

This thesis draws inspiration from both articles. It experiments with news structure-based extraction in its data-processing step and HAC algorithms in its clustering step, as Park et al. did with NewsCube. Yet in the end, for reasons discussed later, the

application provided with this thesis employs Density-Based Spatial Clustering of Applications with Noise (DBSCAN), not HAC. Unlike NewsCube, this thesis is not as interested in finding different political viewpoints within a single language, but rather uncovering the divergent perspectives that become apparent when viewing articles about the same topic in multiple languages. The work by Munson et al. (2013) remains relevant to this thesis because it demonstrated that providing alternative sources of information to a user and telling a user that they are selectively exposing to themselves only one particular viewpoint is not adequate to change that user's behavior, a discouraging outcome that unfortunately this thesis does not attempt to remedy.

Also related to this thesis are works that explore comparing words and documents across languages. Two interesting approaches are explicit semantic analysis through Wikipedia's interlanguage links and a multilingual thesaurus for determining similarity such as EUROVOC for European Union (EU) languages.

Utilizing Wikipedia's knowledge graph, Hassan and Mihalcea (2009) were able to infer the semantic relatedness of two words in different languages without taking into account the meaning of the words. Their model uses explicit semantic analysis (ESA). For each word, a vector is constructed containing the number of instances each Wikipedia article contains that word and these concept vectors are then compared using techniques from linear algebra. Hassan and Mihalcea modified the original ESA algorithm to take into consideration that most concept vectors are sparse by using overlap as the similarity metric between vectors rather than traditional approach of cosine similarity. They also use Wikipedia's category graph to promote category-type concepts in their vectors by giving them a higher weight.

To evaluate their model's performance, they had the gold standard of semantic word relatedness (the Miller-Charles and Finkelstein English word-relatedness datasets) translated into other languages. Their model produced outcomes comparable to the state of the art of monolingual similarity metrics for both monolingual and cross-lingual word

pairs. These results are very interesting and while I originally planned to include their multilingual semantic relatedness work into a similarity metric, in the end I did not because speed is an important factor in usability and on the fly analysis of every word in two documents is computationally expensive. Consequently, I used the more traditional approach to document similarity: term frequency/inverse document frequency (TF-IDF). This thesis is also clearly different from Hassan and Mihalcea (2009) because it considers the similarity between articles rather than individual words and does not use Wikipedia as a knowledge source.

Every single official document that the EU produces must be translated into each of its member states' official languages. This is an excellent trove of parallel texts for Natural Language Processing (NLP) researchers. Steinberger, Pouliquen, and Hagman (2002) used one set of EU-created parallel texts in conjunction with the EUROVOC thesaurus, an EU-sponsored multilingual thesaurus, to identify translations and to compute similarity metrics between documents. Most of their work was done across English and Spanish documents, but they note that their methods are applicable to any language pair from EU member states.

The EU manually labels each official document with one or more terms from the EUROVOC thesaurus for indexing purposes, and Steinberger et al. used these labels to statistically associate certain terms in the documents with each of the tagged terms from the thesaurus. They were then able to compute a similarity metric between documents as well as a binary prediction as to whether or not the two documents were direct translations of one another. For English to Spanish, for example, they were able to deduce that two documents were their corresponding translations 88% of the time.

This thesis adopted the concept from Steinberger et al. (2002) that parallel texts are most similar to one another and applied this intuition to parallel articles from the New York Times in Chinese and English. These parallel articles were used as an evaluation metric to determine which syntactic and semantic features from articles are most

predicative of similarity. On the other hand, this thesis does not statistically associate terms in documents with external tags from a thesaurus.

More recently, researchers have written a number of articles on the classification and clustering of topics on Reddit. Reddit, is the world‘s fifth most frequented website, provides forums for different topics, and each topic is known as a subreddit. Two articles investigating subreddits are worth mentioning in particular with regard to this thesis because of their explicit application of NLP to understanding politics and their use of discovering clusters in unclassified data and unique approaches to visualization.

After presidential-candidate Hillary Clinton’s infamous “deplorables” categorization of then-candidate Donald Trump’s supporters during the 2016 US election, people became curious to explore exactly who these Trump supporters were and what opinions they held. In “Dissecting Trump’s Most Rabid Online Following” (Martin, 2017), Martin used Latent Semantic Analysis (LSA) to analyze the activity of users who commented on multiple subreddits. In traditional LSA, word collocations between two different documents are used to determine the similarity of one document to another, usually via calculating the cosine similarity between two documents mapped into a multidimensional coordinate space. In Martin’s article, the collocations were the users who commented in multiple subreddits, and each subreddit was then mapped into a space containing all of the different subreddits. Martin paid attention to the largest pro-Donald Trump subreddit, `/r/the_donald`, and on those subreddits which users who frequented `/r/the_donald` commented. Then, Martin used simple linear algebra operations, such as adding and subtracting the vectors created by each subreddit, to obtain insights into which subreddits are similar and which are different. Like Martin’s research, this thesis used concepts from linear algebra to determine the similarity of two articles. One prominent area where this thesis diverges from Martin’s approach is that it does not use explicit LSA with collocations, although it explored (but rejected) the similar concept of expanding a document’s word vector with synonyms of each word.

About a month after “Dissecting Trump’s Most Rabid Online Following” was published, McInne published an article that explored different methods of clustering subreddits based on the collocation data from Martin’s work (McInne, 2017). His research explored what a graph of the 10,000 most popular subreddits would look like in two dimensions. To do this, he created a popularity ranking by adding up the magnitude of collocation data for a reddit prior to normalization. Then, he demonstrated how to use SciPy’s sparse matrix data structure to create a large 2D-matrix of collocation data, with the value of a location in the matrix being the number of co-commentators two subreddits have with one another. Since the dimensionality of each vector is still quite high, McInne used Singular Value Decomposition, a linear, slightly-lossy compression technique, to reduce the matrix to 500 dimensions. Then, to visualize the data in two dimensions, he employed the LargeVis non-linear dimensionality reduction technique. Next, the subreddits were clustered using the Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) algorithm. Finally, he covered how he created both interactive and static visualizations of the clusters. This thesis borrowed some concepts from McInne, most notably the use of a density-based clustering algorithm because the number of clusters is unknown a priori and the set of new articles contains noise.

Chapter 3

The Web Application

3.1 Software Development Requirements

This section presents the design, functional, and performance requirements for the application developed in this thesis. It describes these requirements in imprecise language of the type generally avoided in technical documentation or Request for Comments but typical of the software industry's product owner-created requirements documentation.

3.1.1 The Target Audience

When developing any software application, it is important to have a clear vision of the anticipated user group. For this application, the anticipated users have an average level of technical competence. They use the application in one of two ways: for casual browsing or for research. They are American or Chinese and have a passing ability in the foreign language. They prefer to read articles in their native tongue. They probably study, or have studied, international relations, Asian studies, or American studies.

The casual browser is motivated primary by curiosity. He may find coverage of international news repetitive or formulaic and have an intuition that the foreign press reports on topics quite differently. Currently, he isn't very adept at finding foreign

language articles but searches for them from time to time. This application fulfills his need to locate quality articles without having to go through the trouble of trying to find articles from a foreign press. After being introduced to this application, he uses it infrequently as it is not his primary source of news. His visits to the website correspond to those times that there's a noteworthy international affair involving China and the US.

Another possible user is the researcher. The researcher works for the government, a think tank, or academia. If he works for the government, he may be tasked with compiling or analyzing foreign press to brief bureaucrats in the state department or foreign ministry, or for one of the many intelligence gathering agencies. His current process of finding these articles is to visit a preset list of foreign news sites and manually search through the headlines to find those stories relevant to international affairs. After being introduced to this web application, he finds it more convenient than his previous work process and uses it on a daily basis. The application helps him find stories so he can compile a report on how the foreign press is covering a particular issue. If he works for a think-tank or academia, he is probably doing research on an ongoing, long-term issue that comes up in the media quite frequently. He uses the application to keep an eye out for when articles about this issue are published.

3.1.2 Visual/Design Requirements

The application's information hierarchy and visual design must assist the causal browser or researcher accomplish their respective tasks. To that end, the presentation of the application must be similar in style to a typical news sites. On any page, the user must be able to change the application's language to either English or Chinese. Article headlines must must all be in the same language and correspond to the presentation language of the website. Each headline should have a visual indicator of what the article's original language is.

The home page of the application should have stories from the last four days

composed into their respective clusters. Each cluster should display at most five stories, though if there are more stories in the cluster the user must be able to expand the cluster to see them all. Each cluster of stories should be organized in a block, similar to Google News. Clicking on a story should link either directly to the source or to a Google Translate version of the original article translated into the user's chosen language.

There must be a way on the homepage for the user to modify the date. Upon changing the date, the interface presented to the user should be the same as the home page but present articles published around that date. The user should always be able to navigate back in the application by using the browser's back button.

3.1.3 Functional Requirements

To support the use cases described above, certain functional requirements must be met. These include presenting articles from multiple English and Chinese sources, updating the site regularly with new articles, localizing the content to the user's language, and making the website publicly accessible at a well-known URL. Each of these functional requirements is further explained below.

The application must aggregate news from a variety of English and Chinese sources. Both Munson et al. (2013) and Park et al. (2009) note how any one particular news source is going to have its bias. Therefore the more news sources, the better. At a minimum, there should be at least three news sources for each language included in the application. Since news is published regularly and published news changes frequently, news websites should be scraped on a frequent interval, at a minimum once a day.

As the application is meant for both Chinese and American users, it must be localized and internationalized into simplified Chinese and American English. The user must be able to select one of the two display languages at the top-right of the application. All text must then be displayed in the chosen language. Dates and times must likewise be in a format corresponding to the chosen language. Upon visiting the website for the first

time, the application should attempt to determine the user's preferred language by inspecting the `Accept-Language` HTTP header. If the user manually changes the display language while visiting the application, that selection must persist across multiple sessions.

Lastly, as a web application's usefulness is limited by the ability of a user to find it, the application must be available at a well-known, static URL. For this application, the homepage must be located at:

```
http://www.plaintexttransmissions.org/news
```

The final requirements are in regards to performance. Since the application is always online and articles are added to its collection on a regular basis, it is infeasible to pre-cluster articles. Therefore, the daily clustering algorithm must perform well enough to execute on demand whenever a user visits the application. The speed of clustering 100 articles should be less than 200ms, or common attention diversion techniques, such as loading icons, should be presented so that the user perceives the application as fast. Additionally, as the application gathers and processes articles on a regular basis, the time required to gather the articles should never exceed the interval between its invocations.

3.2 Users Guide

This short guide explains how a user with a web browser can use the application.

Visiting the web application

The web application can be browsed using an up-to-date version of Google Chrome, Chromium, Apple Safari, or Mozilla Firefox. The homepage's address to type into the browser is:

```
http://www.plaintexttransmissions.org/news
```

Browsing the latest articles

After navigating to the web application's internet address, the homepage will appear as in Figure 3.1:

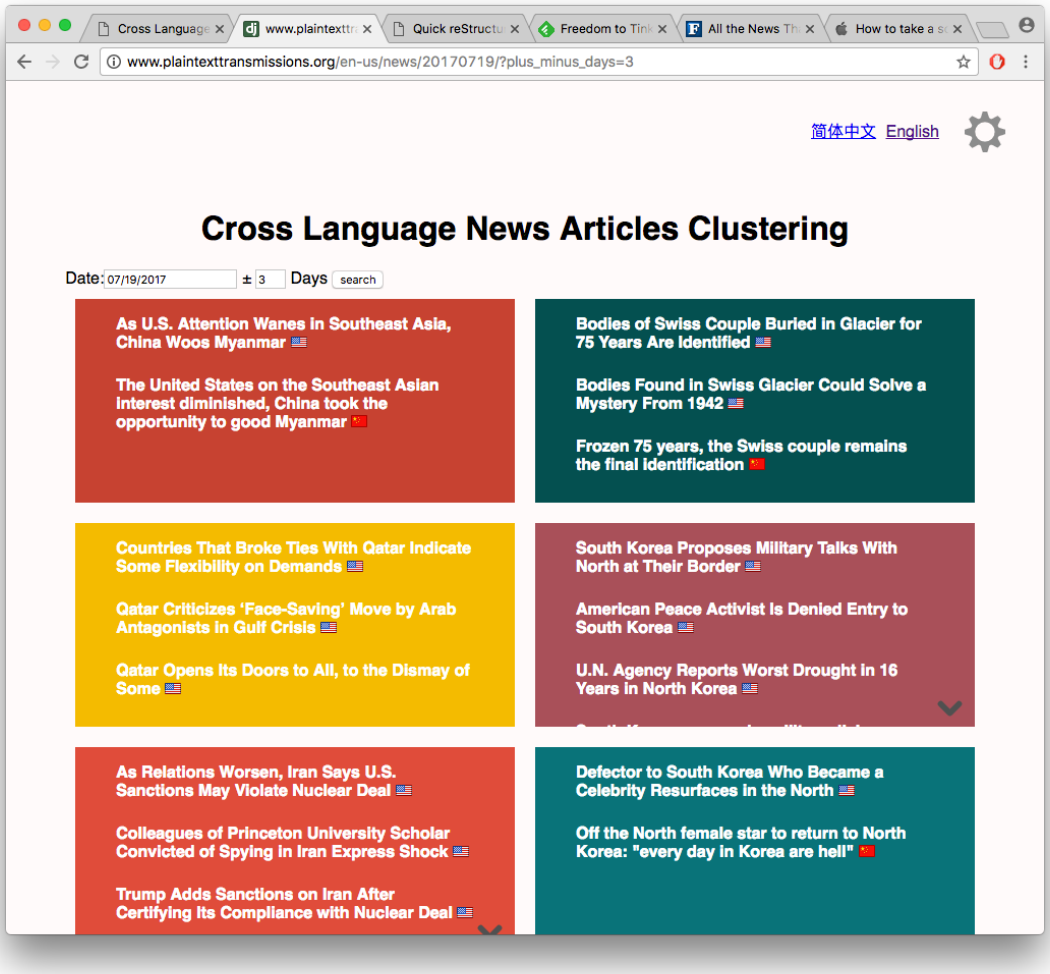

Figure 3.1: The homepage upon first visiting the website

The homepage defaults to the current date with an added date range of plus/minus three days, meaning that by default the application will display clusters from the last four days. The application will attempt to guess the correct user interface language based on the browser and if it guesses wrong the interface's language can be updated at the top-right of the screen. Next to the language selection is a cog which, when selected, displays the

advanced settings.

Each of the colored rectangles in Figure 3.1 is a "story card." A story card is a group of articles about the same topic. Each story card contains, at a minimum, two articles about the same topic. Each card displays the first few articles in a cluster, and if there are more the arrow in the bottom right can be clicked to expand the card and show all of the articles. Figure 3.2 shows an expanded card:

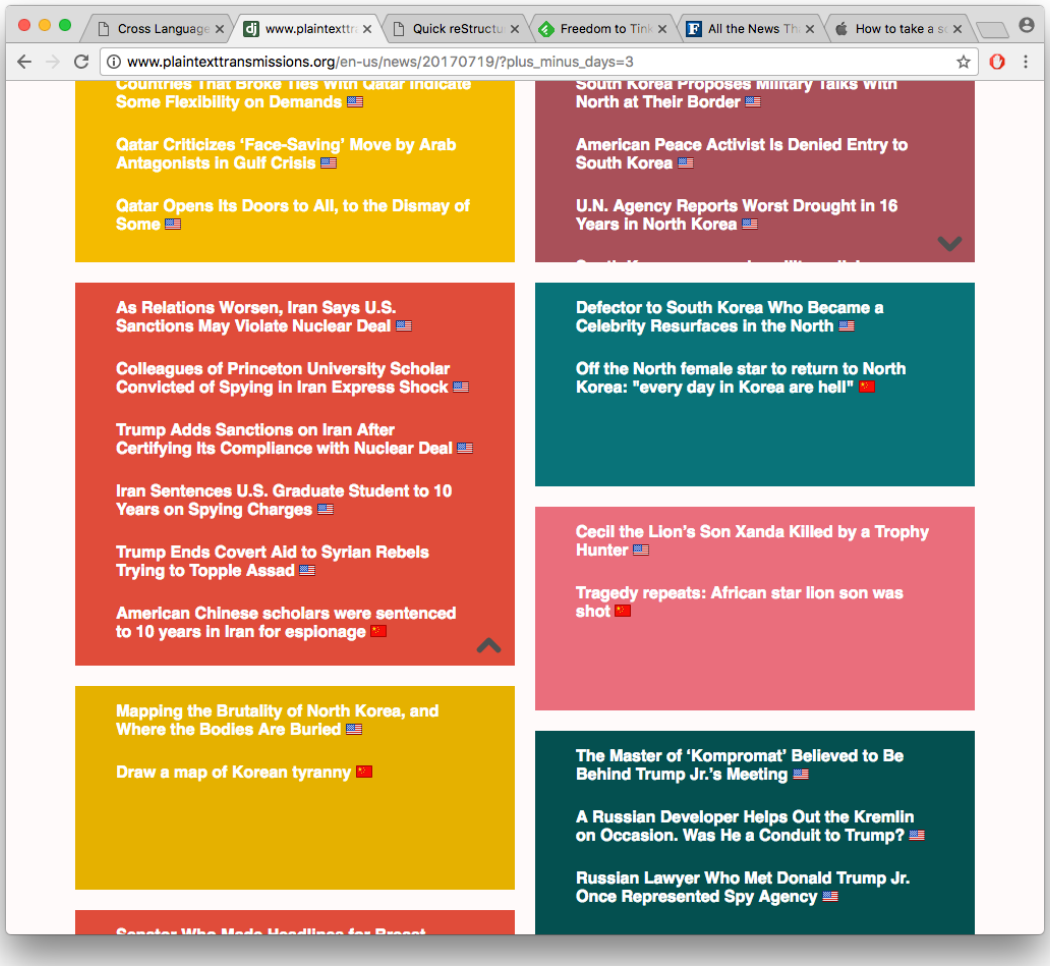

Figure 3.2: Clicking on the disclosure icon in the lower-left expands the story card

Changing the date

The date at the top-left of the screen is interactive and can be modified by clicking on it and selecting another date. Clicking off of the pop over saves the selection and the page can be reloaded with the new clustering date by clicking the search button.

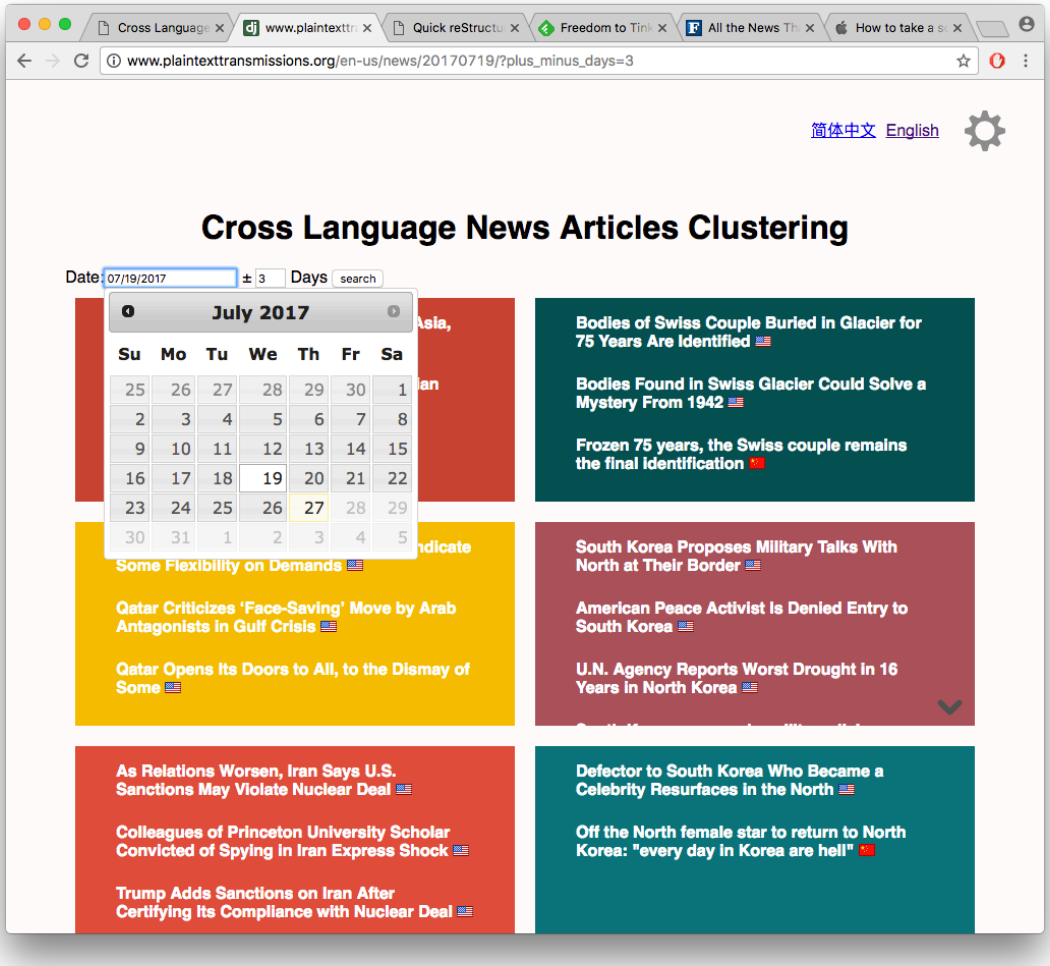

Figure 3.3: Clicking on the calendar allows selecting the center date for clustering the articles

Changing the date range for the story card generation

By default, the story cards are generated for a four-day date range starting four days ago up to the present. In certain cases, it is informative to narrow the date range to only one or two days. It is possible to narrow the date range of the articles used for story card generation from seven days (plus/minus three days) to one day (plus/minus zero days). Clicking the up and down icons in the plus/minus stepper changes the number of days before and after the central date to generate clusters from. To then search with the new date range, click the search button.

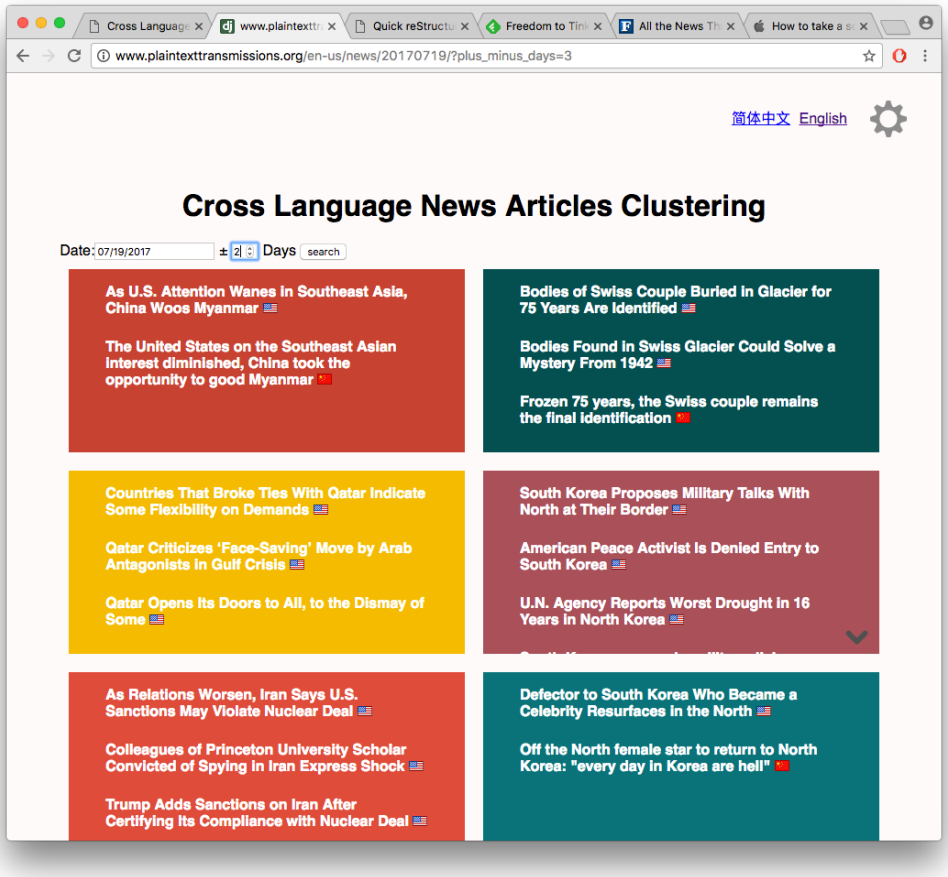

Figure 3.4: Expanding or constricting the number of days around the center state for clustering

Changing the web application's language

The application's interface defaults to either English or Chinese. It attempts to choose the correct one for each user by detecting a preference in the `Accept-Language` HTTP header. If it does not detect a preference, or if the user prefers to use the site in a language that is not the browser's default, then it is possible to change the language by clicking on the preferred language at the top-right of the screen. If the preferred language is changed, it is retained across visits to the application by storing a cookie in the browser.

Including/Excluding news sources

All news sources are included in the story card generation by default. To exclude certain news sources, click on the advanced settings cog at the top-right of the screen and modify the news source selection. Clicking off of the advanced settings pop over saves the selection and reloads the page if the settings have changed.

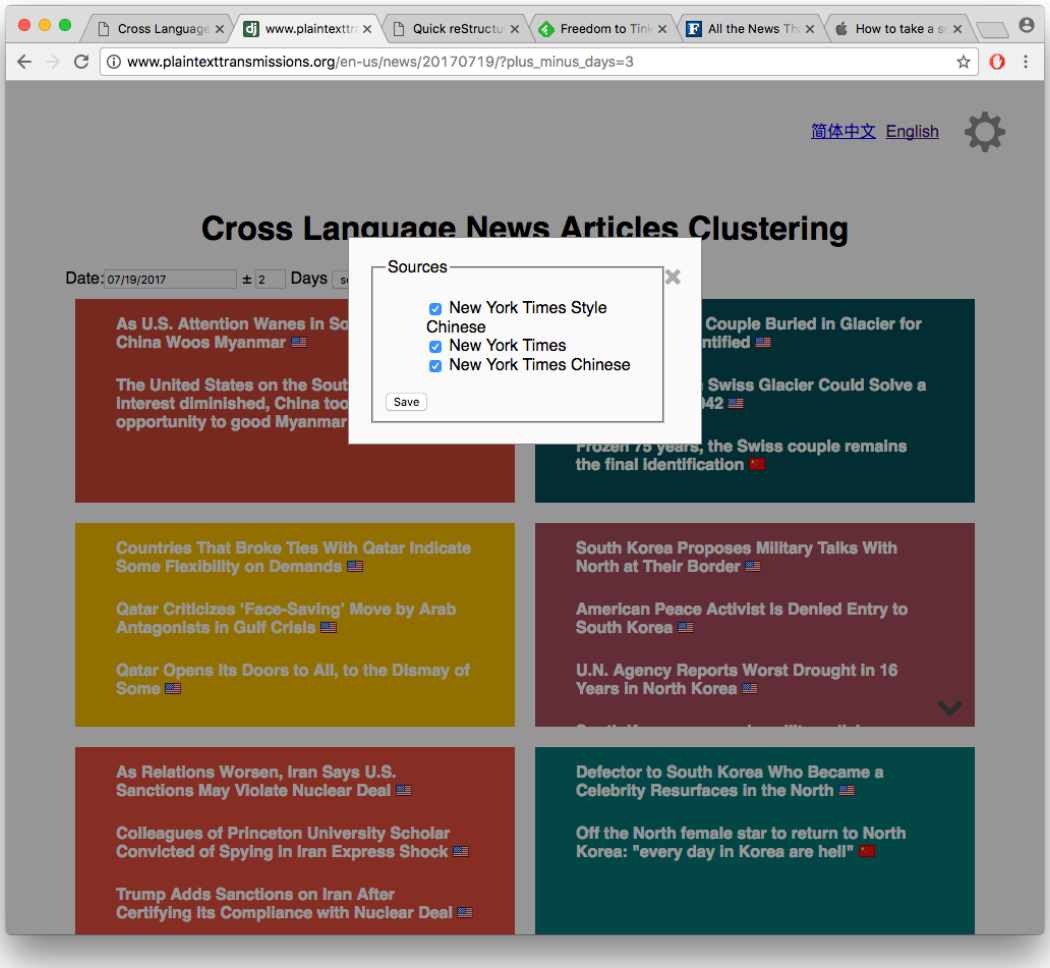

Figure 3.5: Clicking on the cog opens the advanced settings menu

Chapter 4

Implementing Cross-Language News Article Clustering

4.1 Architectural Overview

There are two main components of the application: the service that collects and processes news articles from the web and the web application that clusters the articles, localizes them for the intended audience, and presents them to the user. The entity that links the two components together is a shared PostgreSQL database. A diagram of the entire system can be seen in Figure 4.1.

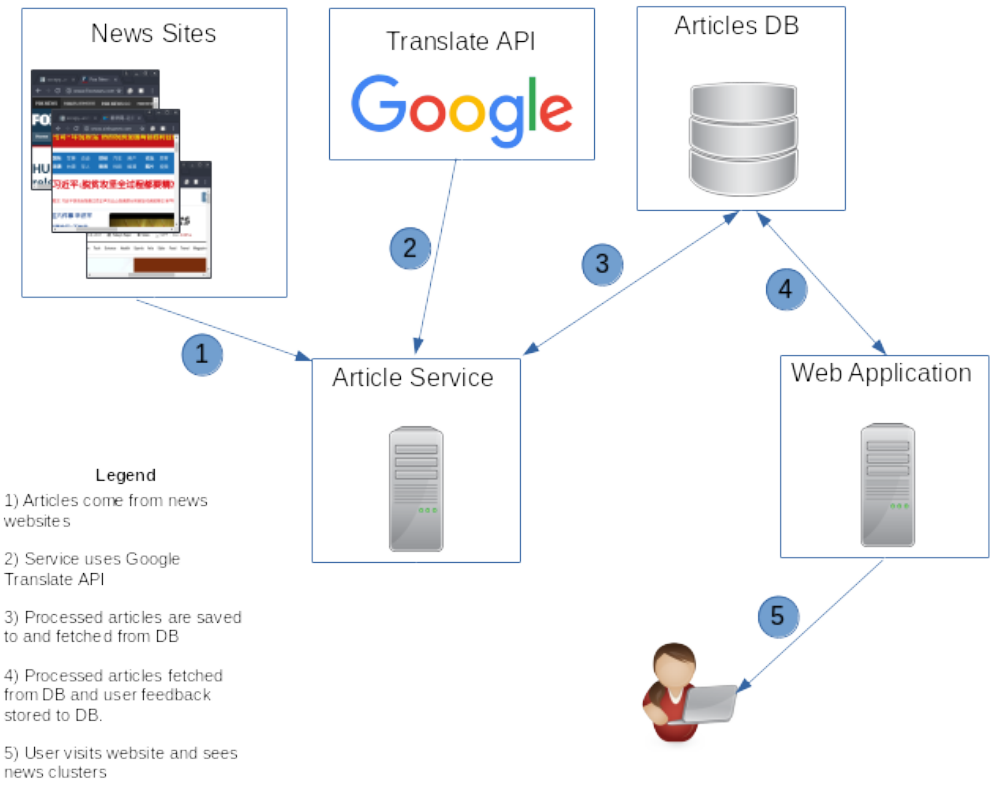

Figure 4.1: A high-level overview of the system architecture

4.1.1 The Article Service

The article service is written in Python 3.6 and deployed on a Fedora Linux server. This same server also has an installation of Stanford CoreNLP, used for annotating and tagging English and Chinese texts. A cron job, which is a task that one can schedule an operating system to execute on a set interval, runs the article service daily. The service's first stage scrapes Chinese and English sources for new or recently changed articles. After fetching an article's text, the service extracts the text from the article and removes any structured data markup. The output of the text processing stage is saved to a PostgreSQL database. The first stage ends when all articles have been fetched and saved. The second stage is text analysis, where each article's text is sent to the Stanford CoreNLP program to be tokenized, lemmatized, and have its tokens' part of speech tagged and named entities

identified. The output of this analysis is saved alongside the article's text in the database. The article service's third and final stage is to look for each Chinese article that has not yet had its tokens and headlines translated, translate these tokens and headlines into English, and save the results back into the database.

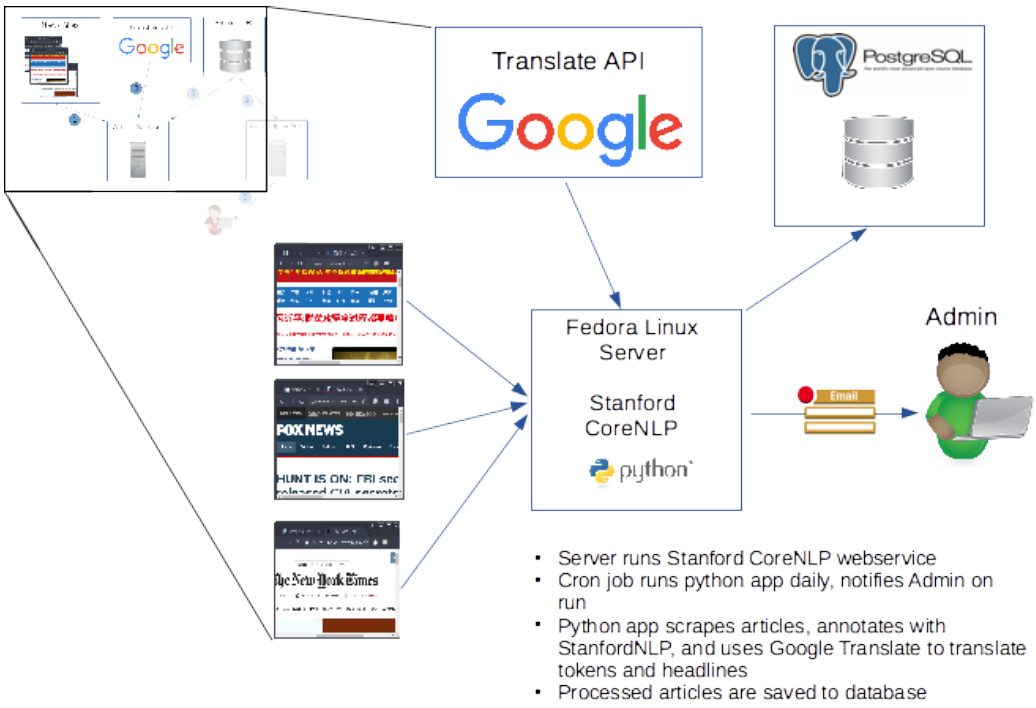

Figure 4.2: The article service's architecture

4.1.2 The Web Application

As noted above, the web application runs on a Fedora Linux server hosted in Amazon Web Services (AWS). It is served using Gunicorn, a Web Server Gateway Interface (WSGI)-compliant pure-Python application server. Nginx is placed in front of Gunicorn and serves static resources, such as images, javascript and css files, and pure HTML files. Requested URLs whose path's first component is "static" are handled this way. Requests for other URLs are proxied by Nginx to the Gunicorn application server. Nginx and Gunicorn communicate with each other over a standard Unix-domain socket.

When a user navigates to the web application's home page, the application clusters

the articles for the current date. Every query to the web application is made in the context of a query date, defaulting to the current date. The query date is the center of a user-specified date range, which defaults to three days before and after the query date and constrains the timespan of articles to be clustered. When a query is made, the application fetches the articles published within that date range and clusters them. It also provides a localized and internationalized user interface by using a combination of Django's built-in tools (see section 4.2.2), GNU gettext, and the translated versions of headlines previously saved into the database.

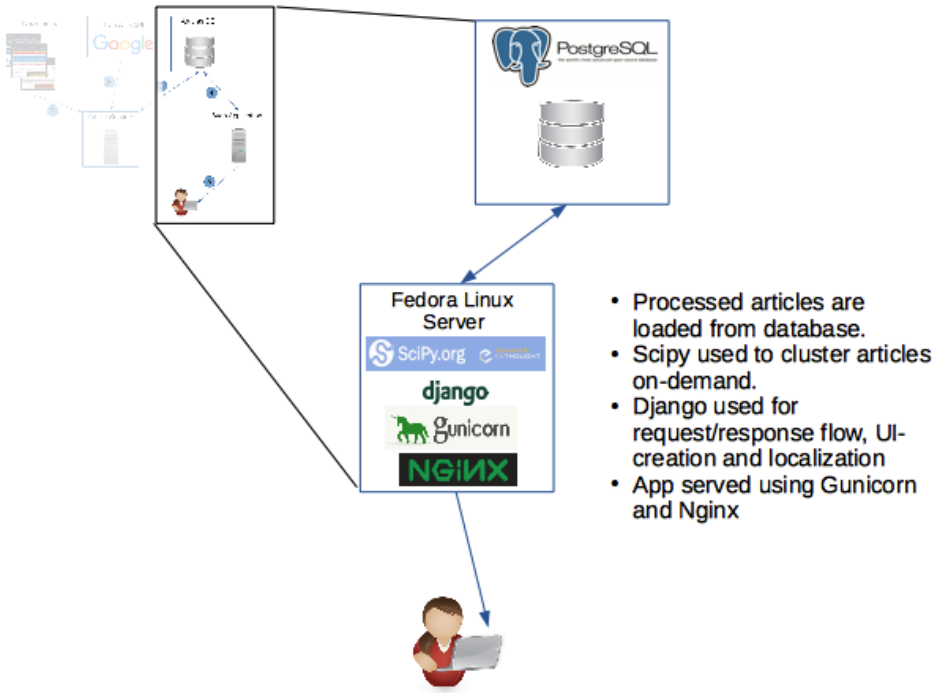

Figure 4.3: The web application's architecture

4.1.3 The Database

An AWS-based, Fedora Linux server has an installation of PostgreSQL (see section 4.2.3). The database is shared between the article service and web application.

4.2 Technology Options and Choices

There are a number of different libraries and frameworks that the application uses to meet its requirements. I've divided the technology choices below into the subsystem they're used in: the article service, web application, and database. Some, like Python 3.6, are used across multiple subsystems. In such cases they are listed in the first subsystem in which they appear.

4.2.1 For the Article Service

CPython 3.6

The core of the article service's logic is written in Python targeting the 3.6 CPython runtime. There were a few different options I considered when it came time to choosing an implementation language. I first looked at what languages were being used in NLP applications and found that many of them were written in Java, Python, and C++. Anticipating that I would rely on code libraries and frameworks written by others, the easiest and most straightforward way to integrate their code would be to use the same programming language these frameworks used. By doing so, I could simply link to them during compilation (C++) or load them at runtime (Java, Python). Additionally, in order to increase my own productivity, the implementation language choice needs to intersect with the set of programming languages I am fluent in. This eliminated C++ from the list of possible choices.

As NLP applications tend to be expensive in both time and space, my initial inclination was to write the application in Java. OpenJDK's implementation of the HotSpot Virtual Machine (VM) is highly performant: just-in-time compilation helps programs run fast, its generational garbage collection manages memory efficiently, and the VM has none of the constraints regarding parallelism that Python has with with its

global interpreter lock (GIL). Additionally, the Stanford CoreNLP library and many web application frameworks are written in Java, so integration with these programs would be trivial if I implemented the article service in Java.

Yet the more research I did, the more I found that most practitioners of the machine-learning/data-science subfields, as well as many academics and researchers working on NLP, use Python. Choosing Python would allow me to leverage the tools that they use, such as the venerable numpy (Walt, Colbert, & Varoquaux, 2011), SciPy (Jones, Oliphant, Peterson, et al., 2001), and Pandas libraries (McKinney, 2010), and learn from the tutorials and articles these practitioners and researchers publish. Additionally, my concerns about Python's performance with regards to the GIL and the fact that it's an interpreted language have been somewhat alleviated by the fact that many of the CPU-intensive algorithms needed to carry out NLP tasks are delegated to compiled native C extensions. These extensions have no restrictions around threading or locking other than those placed on them by good programming practices.

Stanford CoreNLP

One of the steps that Evans (2005) employs to identify similarity in a multilingual text corpus is feature extraction based on different components of the text. These components include parts-of-speech, named entities, dates, and WordNet synsets. As this application must be able to process both English and Chinese texts, I had to find a library capable of performing these tasks for both languages. The natural choice was the Stanford CoreNLP software package by Manning et al. (2014). CoreNLP is written in Java and many of its lower-level NLP functions are accessible solely through a Java interface. Lucky for me, it can also be run as a web application that exposes a high-level NLP tasks by passing JSON messages over HTTP-transport. These tasks, such as tokenization, lemmatization, named entity recognition, and part-of-speech tagging, are sufficient for this application. The article service takes responsibility in managing the life cycle of the

CoreNLP service. It starts CoreNLP up after the service has scraped the articles and is about the begin the analysis phase and terminates it when the service is complete.

Scrapy

Scrapy is “an application framework for crawling web sites and extracting structured data” (Hoffman et al., 2017). There are a couple of different approaches that can be used to gather text from news articles. The first and easiest is to interact with a news source’s application programming interface (API), if one is available. Sites like the New York Times and CNN provide APIs for developers to access part of their published content fairly easily. Some of these APIs do not provide all of an article’s text, in which case the article service could interact with the API to uncover metadata such as where the full text of the article is located and when it was published. A second option is to simply use a HTTP library and write a one-off program that downloads articles from a news source. Once HTTP content is retrieved, a XML parsing library can be used to extract the desired content. This approach is the most customizable and can be used with any news source that publishes content on the web, no matter the structure of that content. A third option is to use a web scraping framework, a choice which on one hand imposes restrictions but on the other can make many tasks trivial to implement.

The approach that the article service uses is a mix of direct API access and scraping framework. After consulting with Ryan Mitchell, Harvard Extension School alumni and author of *Web Scraping with Python: Collecting Data from the Modern Web* who recommended Scrapy, I decided to center the scraping process around Scrapy (Mitchell, 2015; Mitchell, personal communication, January 27, 2017). Scrapy comes “batteries included,” meaning many of tasks commonly encountered while extracting content from the web are already implemented (to a certain degree). For example, Scrapy provides a hook that the application integrates with in order to save each article it downloads into the PostgreSQL database. It also allows one to fill out login forms trivially, which is necessary

in cases where news sites are blocked by paywalls. Most beneficially, the marginal cost of adding news sources is low when compared to a completely custom solution.

Psycopg2

Psycopg2 is a Python wrapper around the `libpq` PostgreSQL client library written in C. It implements the Python Database API, Version 2 (Lemburg, 1999), which is a Python standards document defining an API that different Python database access modules are encouraged to implement. While there are a few different Python PostgreSQL client libraries (see Fuxjäger (2017) for additional possibilities), psycopg2's adoption by large Python frameworks such as Django has made it the de-facto standard PostgreSQL client library for Python.

Google APIs Python Client

There are many different options for machine translation of Chinese to English text. One could choose a service provided on the web or install software that provides machine translations on a local computer. As the quality of the output of statistical machine translation systems is often proportional to the number of example texts translational models have been trained on, it should be no surprise that companies such as Google, Microsoft, and IBM are large players in the translation software-as-a-service (SAAS) field.

Since the translation strategy the article services uses is word-by-word translation, a basic machine translation would work fine. Two areas that an online SAAS solution outperforms a simple bilingual dictionary lookup are when it translates names and novel modern vocabulary. As news articles prominently feature names and writing style evolves over time, an online service suits these needs well. Microsoft is currently (as of May 2017) running a promotion with their Cognitive Services translation API, providing two million characters per month for free. The Google Translate API, on the other hand, costs

$20 per million characters. Thinking about the price savings, I first integrated Microsoft Cognitive Services into the article Service. During the integration process it became clear that the Cognitive Services APIs were unwieldy and not-well documented, so I switched to Google Translate and used Google's published Python client for Google APIs. The simplicity of the interface and the fact that it worked "out-of-the-box" made the choice to stay with Google Translate services easy, notwithstanding the higher cost. Targeted per-word translation caching cut down on the cost significantly.

4.2.2 For the Web Application

Numpy

When it comes to numerical processing in Python, numpy has converged as the de-facto standard library. Its best-known feature is its N-dimensional array, constructed in sequential memory space. This is unlike Python's oft-used `List` type, which looks like an array but is actually a linked-list. Python does have a built-in array datatype, but numpy supports multidimensional arrays whereas the Python array datatype does not. The N-dimensional array also provides highly-optimized vector and matrix-based linear algebra operations, as well as random number generation capabilities. Scipy, scikit-learn, and matplotlib are all built on top, use extensively, and/or integrate very well with numpy.

SciPy and scikit-learn

SciPy is built on top of numpy and is a collection of functions and algorithms implemented in Python to provide a toolkit for scientific computation. It is similar to MATLAB in scope and purpose. SciPy itself contains a number of packages offering distinct functionality, such as the `linalg` package for linear algebra and the `fftpack` package for fast fourier transform routines (Jones et al., 2001). Most relevant to this application is its `cluster` package, which provides algorithms for clustering data.

Integrating with SciPy is scikit-learn, a Python package providing a collection of machine-learning algorithms (Pedregosa et al., 2011). Its algorithms can be divided into two high-level groups–supervised and unsupervised learning. In the supervised group of algorithms, scikit-learn provides implementations of both discrete and linear predictors such as Bayesian Regression, Perceptron, and Stochastic Gradient Descent. In the unsupervised category of algorithms, it provides implementations of DBSCAN, Spectral Clustering, K-Means, and both agglomerative and divisive hierarchical clustering. Additionally, scikit-learn provides functions to evaluate the effectiveness of different algorithms as well as functions to pre-process and transform datasets. The web application employed both scikit-learn's implementation of hierarchical and DBSCAN clustering.

Natural Language Toolkit

Natural Language Toolkit (NLTK) is a Python library implementing many frequently used natural language algorithms (Bird, Loper, & Klein, 2009). Its impetus is both pedagogical and practical–pedagogical because it provides an e-book on NLP for students first learning about Python and NLP and practical because many of the algorithms implemented, while not necessarily the most efficient, have real-world applications. It is meant to make NLP more accessible to beginners, to this end it comes pre-packaged with over 50 of the most popular corpora, from inauguration speeches to WordNet. Its built-in functionality includes tokenization, stemming, tagging, parsing, and semantic reasoning. As most of the NLP tasks, including tokenization and stemming, are performed by the article service, NLTK is used only for its WordNet synset expansion capability. A synset is a group of semantically equivalent words. Synset expansion thus returns all of the words in the same synset as an input word. Synsets were explored as one of the inputs to the clustering algorithms.

Matplotlib

Matplotlib is a Python library that excels at drawing 2D graphs (Hunter, 2007). It integrates with multiple graphic backends and can output various types of images, from PNGs to JPEGs to PostScript files. Most Python developers creating 2D graphs use Matplotlib, and non-developers used to using MATLAB find Matplotlib's API familiar (the original author strived to emulate MATLAB's API). All of the graphs and plots included in this thesis, for example, were created using Matplotlib.

There are a few other Python plotting libraries available, each bringing with them different strengths and weaknesses. Two in particular could have worked for this application are Plotly and Bokeh. Both are interactive and render their graphs in the browser. Plotly is run on a SAAS model and charges each time one of its graphs is rendered. Bokeh, on the other hand, is open source and licensed under BSD 3. I didn't use either of these options because, having been around a long time, Matplotlib is the de-facto standard and has excellent documentation, which is particularly useful when working with a new library.

Django Web Application Framework

There are many different Python web frameworks to choose from, the most notable being Django, Flask, and Pyramid. Brown (2015) from RedHat has written a good comparison of these on his website, which I used when deciding which web framework to use for this application. In the end, I chose to use Django for this application for a few reasons (Django Software Foundation, 2017) . The first is my familiarity with the framework–I have used it in the past so I am already familiar with many of its APIs. The second is that Django has excellent documentation–the online documentation is comprehensive and kept up-to-date and there are many books about Django (see *Two Scoops of Django: Best Practices for Django 1.8* by D. R. Greenfeld and Greenfeld (2015) for a good example). Lastly, Django has first-class support for PostgreSQL

databases, which both the web application and the article service use. This meant that the article service, which uses psycopg2 to access the database, could continue to do so unchanged by the fact that Django would be using its built-in object-relational mapper (ORM) to access the database.

Gunicorn

The Django Web Application Framework itself provides a development HTTP server, and during the development process most developers use this server for its simplicity. This built-in webserver is not appropriate for production applications because it processes requests serially, meaning that only a single request can be serviced at a time. For production applications, a web server that can handle concurrent requests is needed. Gunicorn and Waitress are two options, but since Gunicorn was the recommended and supported by an industry platform-as-a-service provider Heroku, I felt more assured that it was a strong product.

Gunicorn is a WSGI-compliant Python application server. By default, it services requests using a pre-fork worker model. A pre-fork worker model implies that there is a pool of worker processes available to service HTTP requests, and one central coordinating process receives the requests and doles them out to worker processes. Gunicorn must be run behind a proxy server that buffers requests and responses so that the worker processes are not held up by slow clients. This application uses Gunicorn's default worker type, which is the synchronous, process-based worker. Alternatively it could have used asynchronous workers or thread-based workers, but at this point there is not a compelling use case to do so.

Nginx

Gunicorn recommends using Nginx as a proxy server to process requests. Nginx is thus installed globally on the Fedora Linux server and is managed by systemd. The proxy

server fulfills two important purposes–it buffers slow requests to Gunicorn and serves all of the web application's static content. It's configured to run under a non-privileged user account.

4.2.3 For the Database

PostgreSQL database

There are a plethora of relational database options available, such as MSSQL, PostgreSQL, MariaDB, sqlite3, and others. When considering that the database must be well-supported by Python and the libraries and frameworks this application uses, the set of choices narrowed to MariaDB and PostgreSQL. MSSQL is a Microsoft product and would be a challenge to integrate with the otherwise Linux-based stack. Sqlite3 is interesting for small applications but not an appropriate choice for applications with concurrent writing needs, as the article service's crawler is prone to do. As covered in "Appropriate Uses for SQLite," if the database is separated from its application server by the network, and if there is a need for concurrent writes to the database, then a client-server database such as PostgreSQL will work best (Hipp, 2008). In terms of the decision between MariaDB and PostgreSQL, both would have been fine choices for this application but I chose to use PostgreSQL because of its excellent Python support by the psycopg2 module.

4.2.4 Profilers

An important prerequisite to writing fast and memory-efficient programs is understanding algorithmic complexity analysis and the implications of choosing one particular algorithm over another. This alone, though, in modern software development contexts, is insufficient for understanding the performance of an application. In a majority of instances, software developers rely on libraries and frameworks written by others to implement core components of their applications. Many of the libraries and frameworks

listed previously in this section, for example, are highly complex, performance-critical components for this thesis. Beyond carrying out the time-consuming task of algorithmic complexity analysis for all of the library functions incorporated into an application, developers can use other tools to create performance and execution profiles of their complete program while it executes. After the program's execution is complete, these captured profiles can be used to analyze the program's performance and identify potential areas for improvement. While developing this thesis, I used two tools—each of which provided unique insights into different aspects of the application—to fine-tune performance.

cProfile

There are two main approaches to generating execution profiles for a given program: deterministic profiling and statistical profiling. Deterministic profiling involves recording the time that a function or exception handling routine takes to execute. It usually does this by instrumenting source code with callbacks to functions that record the occurrence of important events. This implies that additional computation is carried out when these events occur, causing significant overhead to the runtime performance of the program under profile. Likewise, statistical profiling affects the speed of the executing program by periodically generating interrupts to record the location of the profiled program's instruction pointer, though the negative runtime impact of statistical profiling is usually only a fraction of that of deterministic profiling. After the completion of the recording process, statistical profiling programs are able to infer how many times a function was executed and how long its execution took based on these captured instruction pointer locations.

The Python Standard Library (STL) includes two deterministic profilers, the `profile` and `cProfile` modules. `profile` is written in Python, and, as its name implies, `cProfile` is written in C. They have similar programming interfaces, so for standard uses

the Python documentation recommends using `cProfile` for its non-interpreted (and quicker) execution. These profilers use three hooks that the Python VM provides to record the times that functions are entered and exited, as well as the times that exception events occur. The Python module `pstats` works in conjunction with `cProfile` to filter, sort, and display a captured execution profile. I used these modules to understand the execution profile—and to identify optimization opportunities—for many of the algorithms in Section 4.4.

Django Debug Toolbar

As one of the premier web application frameworks for Python, Django provides many high-level programming abstractions, such as view templates and an ORM, that make developing dynamic web content simpler and quicker. It is very important that a developer properly understand these abstractions prior to developing a Django-based web application. As is often the case, the higher the level of abstraction the more important it is to understand its correct usage in order to guarantee optimal performance. The improper use of a high-level abstraction often leaves developers unsure as to why their program doesn't work the way they intended it to or performs poorly.

Using the ORM, in particular, has a steep learning curve. It isn't always obvious how to create non-trivial queries using Python syntax, nor is it always clear how these queries will be translated by the ORM into SQL queries and their performance characteristics. One of the most useful functionalities that the Django Debug Toolbar provides is the capture of each ORM query's post-translated, raw SQL prior to it being sent to the database (Django Debug Toolbar developers and contributors, 2017). For example, it might not be evident at first glance that the `fetch_articles` function in Listing 1 generates the SQL in Listing 2, but the Django Debug Toolbar allows the developer to ensure that it does.

```python
def fetch_articles(start_date, end_date)
"""Returns articles and new sources published between start_date and end_date

Parameters:
    start_date: the earlier timezone-aware datetime
    end_date: the later timezone-aware datetime

Returns:
    A queryset with only the articles whose publish date is between
    those two dates, and their associated news sources.
"""
    q = between_dates(start_date, end_date)
    q = q.select_related('news_source')
    return q.values_list('news_source__id', 'news_source__name').distinct()


def between_dates(start_date, end_date):
    # ensure that the dates are timezone-aware
    for d in (start_date, end_date):
        if d.tzinfo is None or d.tzinfo.utcoffset(d) is None:
            raise TypeError("Needs tz-aware datetime")

    return Article.objects.filter(
        pub_date__gte=start_date
    ).filter(
        pub_date__lte=end_date
    )
```

Listing 1: A Python query for the Django ORM

```sql
SELECT ***
FROM "article"
INNER JOIN "news_source" ON ("article"."news_source" = "news_source"."id")
WHERE ("article"."pub_date" >= '2017-05-12T00:00:00-04:00'::timestampz
       AND "article"."pub_date" <= '2017-05-18T00:00:00-04:00'::timestampz
       AND NOT ("article"."corenlp_json" IS NULL))
```

Listing 2: The raw SQL generated by the Django ORM

I used the Django Debug Toolbar extensively during the development of the web application for this thesis to understand how the ORM translated Python queries into SQL and which queries stood out for their positive and negative performance characteristics. An example of one such application is covered in Chapter 5.

4.3 Implementation of Core Components

This section details how different parts of the article service and web application are implemented. In particular it reviews the PostgreSQL database schema, the Python class implemented to provide programmatic access to the database, and the implementations of the article scraper and the Django web application.

4.3.1 The Data Model

There are three tables in the database: `news_source`, `article`, and `headline`:

`news_source`

A news source is a location on the web from which news articles are scraped.

Column	Type
id	integer
domain	character varying(128)
created_on	timestamp without time zone
name	character varying(128)

Table 4.1: The database's `news_source` table

The `news_source` table's primary key is `id`. Its `domain` column holds the root domain of a news source, like cnn.com or nytimes.com. Its `created_on` column holds the date and time on which the news source was added to the database. The name column is a human-readable description of the news source. For example, for cnn.com the name is CNN.

`article`

The `article` table holds scraped articles from a news source.

Column	Type
id	integer
content	text
pub_date	timestamp with time zone
web_url	character varying(1024)
news_source	integer
corenlp_json	jsonb
lang	language enum

Table 4.2: The database's `article` table

Like the `news_source` table, the `article` table's primary key is `id`. The `content` column holds the raw, scraped article text. The `pub_date` column is the article's published-on date, to the best accuracy possible. Some news sources only provide the date and not the time an article is published, and in these cases the `pub_date` for the article is set to midnight EST of that date. `web_url` is the URL from which the article was scraped. `news_source` is a foreign key to the `news_source` table–it is mandatory for each article to have an associated news source. The `corenlp_json` column is nullable. When set, it contains the JSON response of the processing of the article's `content` by the CoreNLP server. Later, if the source article is in Chinese, this column is updated with the translated tokens are added to the JSON. The `lang` column holds the source language of the article. Valid enumeration values for this column are `zh-Hans` and `en`.

`headline`

The `headline` table holds the source and translated language headlines for an article. The headlines are translated when the articles are scraped rather than when they are displayed in the web app for two reasons. First, it separates the performance of the web app from the performance of Google Translate (which has no performance guarantees). Second, the cost of translation is incurred only once rather than each time the web application is displayed. The table schema shown in Table 4.3 is simple:

Column	Type
id	integer
article_id	integer
lang	language enum
headline	text

Table 4.3: The database's `headline` table

The `id` column is the primary key for the table. The `article_id` column holds a reference to which article the translation is for. The `lang` column holds whether the headline is in English or Chinese, and the `headline` column holds the actual text of the headline.

4.3.2 Python Access to the Database

To facilitate uniform access to the database by both the article service and web application, I wrote a Python class called `ArticleDB` that is utilized by each of the two packages. Per the documented best practices in Gregorio and Varrazzo (2016), it is best to share a single psycopg2 `Connection` instance across as many threads as are needed, as the creation of a connection is an expensive operation (often SSL over TCP). With this in mind, the ArticleDB contains a singleton instance of a psycopg2 `Connection` class that

clients use to create cursors. As Listing 3 shows, a Python context manager can be used on the `ArticleDB` class to automatically commit a set of transactions when they execute successfully or roll back the transaction on failure:

```
with db_article as db:  # commits or rollbacks on exiting scope
    db.save_articles(articles)
```

Listing 3: Using the `ArticleDB` to obtain cursors over a shared connection

4.3.3 Scraping the Web for Articles

The article service uses the Scrapy framework as its chassis for scraping. An overview of the system design of the Scrapy framework is shown in Figure 4.4:

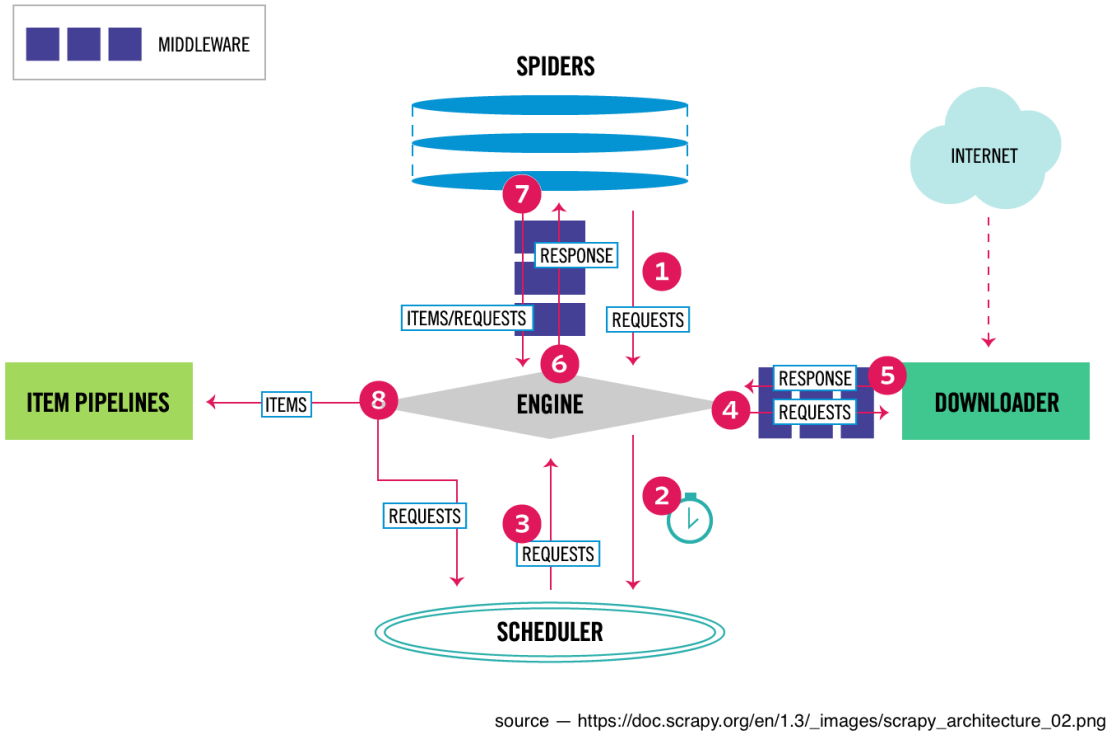

Figure 4.4: Overview of the Scrapy framework architecture

The article service integrates with the Scrapy framework at a few key points. Figure 4.5 uses the same colors and shapes as Figure 4.4 to demonstrate the role certain

classes in the article service play the framework. Notice, for example, that `PostgresSqlPipeline` corresponds to an Item Pipeline in Figure 4.4 because it is a light-green rectangle:

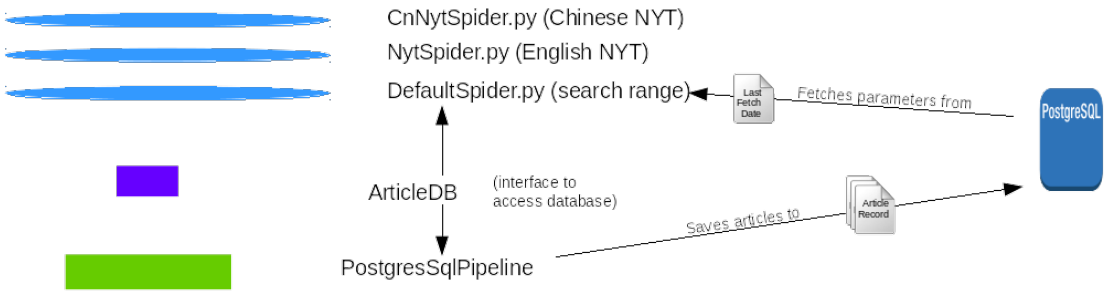

Figure 4.5: The article service's integration with the Scrapy framework

The numbered steps in Figure 4.4 correspond to the items in Figure 4.6:

1. Spiders create HTTP requests.
2. The HTTP requests are passed to a scheduler to schedule.
3. Scheduled requests are returned to the engine.
4. The downloader downloads requests.
5. The downloader returns responses.
6. The spider processes the individual responses.
7. Using the processed response, the spider creates new requests for scheduling or extracting items.
8. Extracted items are passed to the items pipeline for further processing (such as saving to a database).

Figure 4.6: Order of operations in the Scrapy framework

The top-half of the image uses the same colors and shapes from the architecture diagram in the bottom half to demonstrate how each class fits into the framework. Each implementation of a spider that crawls a news source has access to an `ArticleDB` instance, which it uses to obtain the datetime a news source was last crawled. If necessary for sites that require authentication, the spider programmatically logs into the news source. Finally, the spider begins the crawl and steps 1-7 from Figure 4.6 are executed until completion. The `PostgreSqlPipeline` class is plugged in at step 8 and saves each fetched article into the `article` table in the database. An article object produced by Scrapy has attributes for each of the non-nullable columns in the `article` table. Lastly, each of the headlines is translated into English and Chinese and saved to the `headline` table.

4.3.4 Django as a Foundation

The Django Web Application's file structure is put together in a way similar to the Django "Cookie Cutter" template's default layout (D. R. Greenfeld, 2017). The interesting parts are shown below:

```
clds_web/
├── news/
│   ├── admin.py
│   ├── cluster.py
│   ├── models.py
│   ├── tests.py
│   ├── urls.py
│   └── views.py
└── templates/
└── news/
└── overview_list.html
config/
├── settings
│   ├── base.py
│   ├── __init__.py
│   ├── local.py
│   ├── production.py
│   └── test.py
└── wsgi.py

Pipfile
Pipfile.lock
```

Figure 4.7: Important code files in the web application

The `clds_web.news` package is the main Django app responsible for creating the news article clusters and displaying them to the users. In this package, there are a number of interesting files:

- `admin.py`

  This module is used for registering the web application's specific models for the built-in Django Admin project. By doing so, it's possible to use the pre-made

Django Admin web backend to view and modify the `article` and `news_source` tables.

- `cluster.py`

  This module contains a definition of the `Cluster` class that retrieves and clusters articles for a certain date range. The `Cluster` class contains a set of topically-related articles and convenience methods to access the article headlines.

- `models.py`

  This module contains all of the definitions of the different models used in the web application. Most of these classes subclass the `Django.db.models.Model` class so they can inherit the Django Object-Relational Mapper's model management capabilities. This includes CRUD operations that persist and update database state. Other models which cannot be mapped one-for-one with database tables do not inherit from this base class and when they need access to the database use the `ArticleDB`'s connection singleton.

- `urls.py`

  Django uses a url mapping scheme to map requests to view classes. This file contains those mappings. For example:

```
url(
    regex=r'^$',
    view=views.OverviewList.as_view(),
    name='list'
)
```

Listing 4: The Django URL route for the web application's home page

  maps the empty path (specified as `r'^$'`) to the `OverviewList` view.

- `views.py`

This file contains all of the class-based and function-based views in the application. Many of the views' layouts are defined by Django templates, which use the Django Template Language.

- `templates` directory

  The `templates` directory contains all of the Django templates used in each of the Django apps. As this web application only has a single Django app `news`, there is a `news` directory nested in the `templates` directory. Within this directory, there are a number of template files with html extensions, even though the diagram above only shows a single file for brevity.

- `settings` directory

  The Python files in the `config/settings` are used to configure the Django application. `base.py` has settings shared across all environments, such as the choice to use a PostgreSQL backend and layout of the file structure. The `local.py`, `production.py`, and `test.py` either override or add settings to this base configuration. For example, the `local.py` file configures Django Debug Toolbar to be included in the application in order to aid with development debugging needs. The production settings file configures generic error messages, whereas in the local environment it's helpful to have a complete stack trace.

- `wsgi.py`

  This is the entry-point for a WSGI-compliant web server. Both the Django development server and the Gunicorn web server use this module as their entry point into serving the web app.

- `Pipfile` and `Pipfile.lock`

  These files specify the web application's Python package dependencies. `Pipfile` is modified by developers to specify package and version specifiers, and

`Pipfile.lock` is generated by the packaging tool. `Pipfile.lock` has the exact versions of each of the dependencies so that when the application is shared between multiple developers or deployed to production the particular version of a dependency is always the same. These files are used in conjunction with a Python tool called Pipenv, which uses these files to download dependencies and update the Python runtime path so that it uses a virtual environment. As of July 2017, Pipenv is still in beta release, but appears to be on track to replace virtualenvwrapper and standalone pip usage with requirements.txt files.

4.4 Interesting Algorithms

The "story cards" shown by the web application to the user are created using a clustering algorithm. There are many different clustering algorithms one can choose from. The applicability and efficacy of a particular clustering algorithm must be determined on a problem-by-problem basis. This section summarizes a couple of the different algorithms for cross-language news article clustering, namely hierarchical clustering and density-based clustering. It also reviews some of the distance metrics and term weighting algorithms related to classification in NLP. It closes by covering the least-recently used (LRU) caching algorithm used in the article service's translation step and details how the Python standard library implements the `functools.lru_cache` decorator.

4.4.1 Hierarchical Clustering

There are a few characteristics inherent with clustering news articles across different languages that one must take into consideration when selecting the particular clustering algorithm to use. The first is that there are going to be many articles for which there is no other article about the same topic. A local newspaper will publish stories about events that may not be covered by other local newspapers or larger regional newspapers.

This characteristic is even more prevalent across national borders—the vast majority of the set of article topics covered in a Chinese newspaper is disjoint with the set of article topics covered in an American newspaper.

Another characteristic to consider is that news follows current events and what causes particular events to become newsworthy is unpredictable. News organizations usually create different sections such as “Finance,” “Sports,” and “International News,” among others. With a corpus of old news articles and their classification into different sections, it is possible to predict an unseen news article’s section using supervised machine learning algorithms. Yet the topics covered in a section are diverse, and the goal of this thesis is to group articles by topic, not by section. In other words, Chinese ping pong tournaments should not be included in the same cluster as American fencing tournaments, even though they both would be published in the sports section. Consequently, any clustering algorithm for this task needs to have as an input the within-cluster precision of similarity, outside of which articles are not considered to be part of the same cluster.

Hierarchical clustering is a clustering algorithm that places all samples and resultant clusters into a hierarchy based on their distances to one another. There are two approaches to hierarchical clustering, divisive and agglomerative. Divisive hierarchical clustering algorithms start with a single cluster containing all of the samples and proceed through a series of splits until each sample is in its own cluster. On the other hand, agglomerative hierarchical clustering algorithms start by assigning each sample to a different cluster and then, through a series of cluster joins, constructs a hierarchy ending with all samples in a single cluster.

An explicit assumption of hierarchical clustering algorithms is that all samples have some relationship with each other and thus can be placed into a single tree. For example, the scientific consensus is that all forms on life on earth descended from a single ancestor. Recall that the hierarchy in the classic Linnaean taxonomy is, from top to bottom, domain, kingdom, phylum, class, order, family, genus, and species. There are

three main domains of life–Archaea, Bacteria, and Eukarya–in the Woese system of the tree of life (Woese, Kandler, & Wheelis, 1990). If you were to consider that these three domains all are part of the “life” supercluster, then a hierarchical clustering algorithm is a good way to determine how the domains, families, species, and all the myriad forms of life are related to one another. If the problem presented to us is to identify clusters that represent the families in the taxonomies then there needs to be a cut-off point, below which there are only families and above which there are no connections between families. Identifying this exact point below which there is a cluster and above which there is not is a difficult problem in hierarchical clustering and will be further explored after first explaining the basic HAC algorithm.

An implementation of the HAC Algorithm

The high-level algorithmic steps needed to perform HAC on *M* samples, each of *N* dimensions, are show in Figure 4.8:

1. Transform each of M samples into an N-dimensional vector.
2. Create M clusters, each containing a single N-dimensional vector.
3. Compute the distance between each cluster using a distance metric.
4. Take the two clusters which have the least distance between them and join them into a new cluster.
5. Remove the two clusters from step 4 from the list of clusters.
6. If there’s only a single cluster remaining, exit the algorithm.
7. Jump to step 3.

Figure 4.8: Outline of the hierarchical agglomerative clustering algorithm

Figure 4.9 is an example of the algorithm from Figureal 4.8 using a 2D dataset constructed to have three different clusters. The dataset used to create Figure 4.9 is shown in Table 4.4:

Index	X	Y	Cluster
0	11.9	4.4	1
1	10.5	2.3	1
2	11.2	4.4	1
3	10.0	3.9	1
4	11.7	6.6	1
5	2.3	21.3	2
6	3.9	19.7	2
7	4.4	19.7	2
8	2.5	17.4	2
9	1.4	19.1	2
10	16.0	11.5	3
11	14.0	10.4	3
12	13.4	10.7	3
13	13.6	11.1	3
14	14.1	11.1	3

Table 4.4: *X* and *Y* coordinates generating three small clusters used for Figure 4.9

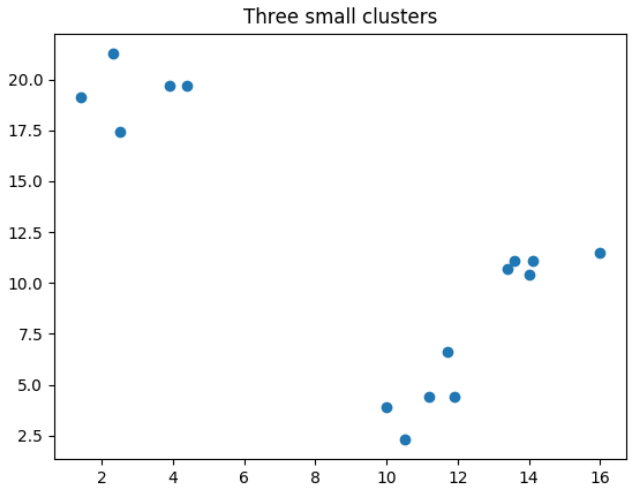

Figure 4.9: Three small clusters scattered in the *XY* coordinate system

Assuming that the dataset above is loaded into a numpy 15-by-2 matrix *X* with each row representing a cluster, the next task is to perform the clustering. Listing 5 on page 52 demonstrates one possible implementation.

```
def hac(X, mean_fn=mean, dist_fn=euclidean_dist):
    """Perform hierarchical agglomerative clustering.

    inputs:
        X - an M-by-2 ndarray
        mean_fn - a function used to calculate the mean between two
            clusters
        distance_fn - a function used to calculate the distance
            between two clusters

    returns:
        X - an M+(M - 1)-by-4 ndarray.  The additional M+1 rows
            capture the the joins (column 3) and the distance between
            clusters used for the joins (column 4)

        merges - a list of lists in the format 'a b c' where a
            is the index of cluster 1, b the index of cluster 2, and
            c the index of the new cluster they form when merged.
        """


    CLUSTER_IDX, DIST_IDX = 2, 3
    X = np.append(X, np.zeros(X.shape), axis=1)
    mask = np.ones(len(X), dtype=bool)
    merges = []

    while len(X[mask]) > 1:                                    # Step 6
        new_cluster_idx = len(X)

        d_mat = dist_fn(X[mask][:,:2])                         # Step 3
        min_d = [*np.unravel_index(np.nanargmin(d_mat), d_mat.shape)]
        shifts = [shift(mask, min_d[0]), shift(mask, min_d[1])]
        X = np.append(X, [mean_fn(X[shifts])], axis=0)         # Step 4
        mask[shifts] = False                                   # Step 5

        X[shifts, DIST_IDX] = np.full(2, d_mat[min_d[0]][min_d[1]])
        X[shifts, CLUSTER_IDX] = new_cluster_idx
        mask = np.append(mask, [True])
        merges.append(shifts + [new_cluster_idx])

    return X, merges
```

Listing 5: Function that performs HAC

Since the `mask` array is used to mask off rows that have already been processed from being input to the distance matrix creation, the `shift` function shown in Listing 6 is needed to recover the indices in the original matrix `X`:

```
def shift(mask, idx):
    """Using the mask, recovers the original index value
    from the unmasked array.

    inputs:
        mask - a bool array
        idx - the index to adjust

    returns:
        the adjusted index
    """
    left = idx
    for m in mask:
        if not m:
            idx += 1
            continue
        if left == 0:
            return idx
        left -= 1
```

Listing 6: An implementation of the shift function

A `mean_fn` can be passed as a parameter to the `hac` function so that the caller can specify which type of linkage algorithm to use. For example, in calculating the distance between two clusters, each with multiple samples, the cluster-to-cluster distance can be calculated from those samples nearest to each other, furthest from each other, or the average of all samples. Likewise, a reference to a distance metric function can be passed as the `dist_fn` argument so the caller can specify which type of pairwise distance metric, such as the Manhattan or Euclidean distance metrics, to use.

Listing 7 is an implementation of the default Euclidean distance metric:

```
def euclidean_dist(X):
    """Calculates the euclidean distance between each point.

    inputs:
        X - an M-by-2 matrix

    returns:
        an M-by-M distance matrix with the upper-right
        triangle filled in with pairwise distances.  The lower
        triangle and the identity diagonal are filled with
        np.nan.
    """
    dim = len(X)
    dist_mat = np.full((dim, dim), np.nan)

    for i in range(dim):
        for j in range(i + 1, dim):
            dist_mat[i, j] = np.linalg.norm(X[i,] - X[j,])
    return dist_mat
```

Listing 7: A function for calculating the Euclidean distance between two vectors

While this is the basic outline of the algorithm, SciPy's `scipy.cluster.hierarchy` package provides optimized routines that are written for and compiled with Cython, an optimizing static compiler for Python that improves Python performance when c-type declarations are added to Python source and also makes calling C and C++ libraries with Python trivial. For example, one capability that the package provides is drawing dendrograms. Figure 4.10 is a visualization of the three small clusters as a dendrogram:

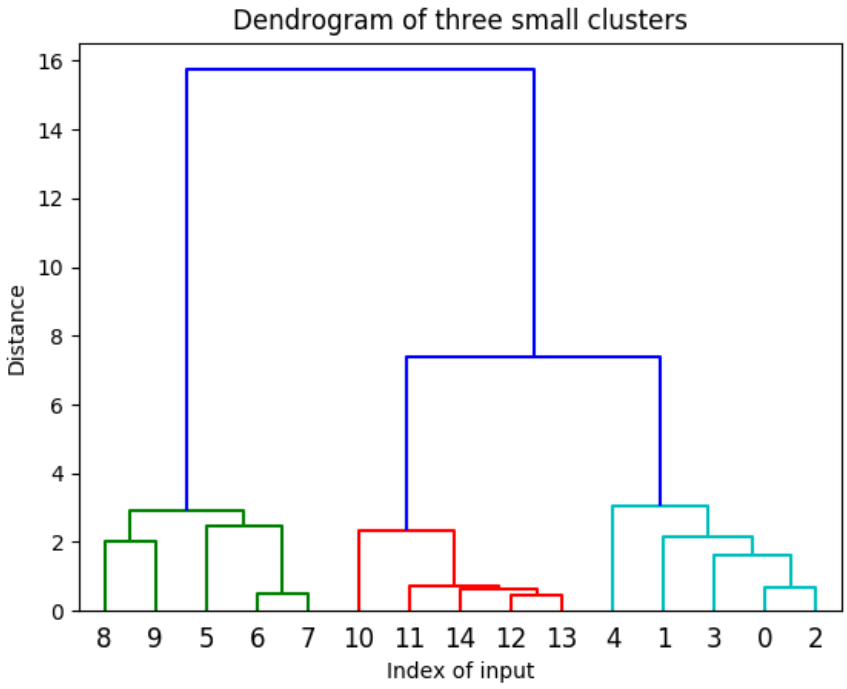

Figure 4.10: Dendrogram produced by hierarchical agglomerative clustering. Note that the green nodes correspond to cluster 2, the red nodes to cluster 3, and the teal nodes to cluster 1 in Table 4.4.

Determining the number of clusters: the inconsistency method

As alluded to in the introduction to hierarchical clustering, finding the cutoff point in hierarchical clustering that chops off the root of the tree and asserts that the model predicts there are a certain number of distinct clusters in the population is difficult (Hees, 2015). There are two methods that can be used to try to determine the number of clusters when using HAC: the inconsistency method and the elbow method. The inconsistency method uses the height of a merge `h`, the average height of merges under the merge `avg`, and the standard deviation of the merge heights under the merge `std` in the following equation to determine an inconsistency metric:

$$inconsistency = \frac{h - avg}{std}$$

The higher the inconsistency value of a cluster the more dissimilar its contents are, indicating perhaps that its contents are actually different clusters. SciPy provides an implementation of the inconsistency function that requires a depth specification as input. This is used for calculating the `avg` and `std` values. The `scipy.cluster.hierarchy.inconsistent` function has two parameters, the linkage matrix `Z` created by one of the many linkage family of functions in `scipy.cluster.hierarchy` and a depth `d`, with a default value of 2. When executed on the input data from Table 4.4, the inconsistency values computed for all the merges are show in Table 4.5 (from root to leaves).

Avg	Std	# Links	Inconsistency
8.69	6.5	3	1.08
4.26	2.75	3	1.15
2.58	0.63	2	0.71
2.47	0.45	3	1.02
1.48	1.38	2	0.71
1.54	1.14	2	0.71
1.89	0.36	2	0.71
2.02	0	1	0
1.17	0.66	2	0.71
0.69	0.05	2	0.71
0.7	0	1	0
0.55	0.15	2	0.71
0.5	0	1	0
0.45	0	1	0

Table 4.5: Intra-cluster inconsistency of three small clusters

The challenge is in determining the correct inconsistency value to split at. Looking at the data above, one might say that a value of 1 looks like it could be a good place to split the results. Unfortunately, doing so would result in four clusters, not the correct value of three clusters. The root cause of the problem is that the samples have different variances within their clusters. Some clusters have their samples distributed more densely than others, such as the rightmost cluster compared to the top-left cluster in Figure 4.9. Moreover, the inconsistency metric varies with the data its used on, so automatically

determining an inconsistency value to split at is a bit of guess-and-check and I do not attempt to employ this method while clustering the news articles.

Determining the number of clusters: the elbow method

The elbow method is the second method that can be used to automatically determine the distinct clusters in a hierarchical clustering. The idea behind the elbow method is to find the point in the hierarchical clustering at which the change in the distance between neighbor points increases most rapidly. In other words, the elbow method uses the global maximum of the second derivative of the merge distances as the number of clusters.

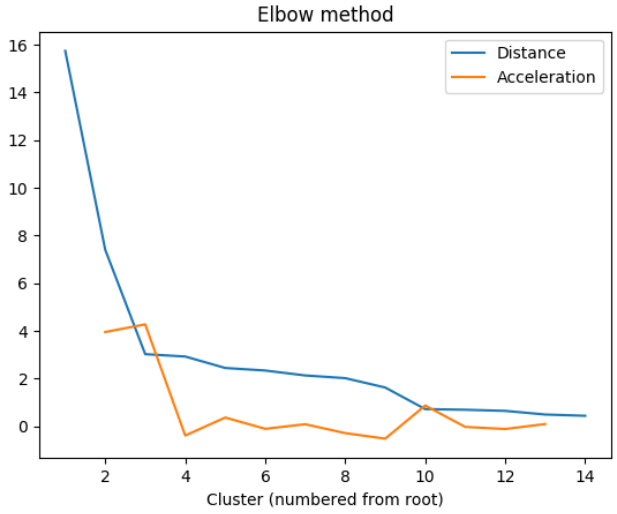

Figure 4.11: Acceleration used by the elbow method to determine number of clusters

Figure 4.11 shows that for our input data the rate of change of the distances increases most rapidly at 3, which also happens to be the number of pre-determined clusters this example was created with. In this case, the elbow method is able to accurately infer the correct cluster count—but in many cases it cannot. Having a similar weakness with the inconsistency method, the elbow method does not perform well when different clusters have different intracluster variance (Hees, 2015). Additionally, by nature of its

design, the elbow method is not able to identify cases where there is actually only a single cluster. For these reasons, as well as the fact that hierarchical clustering is not designed to account for noise, the web application does not use hierarchical clustering to cluster news articles.

4.4.2 DBSCAN

DBSCAN is a clustering algorithm that can discover clusters of arbitrary shape. According to Ester, Kriegel, Sander, Xu, et al. (1996), DBSCAN has improvements over the then state-of-the-art density-based clustering algorithm, CLARANS. It works particularly well when precise knowledge of what data will be clustered is lacking, which is an apt description of clustering news articles because news topics are continuously changing.

The theory behind DBSCAN is intuitive. Imagine the samples to cluster each to have N-features, meaning they can be charted in N-dimensional space. Using a distance metric, it's possible to measure the distance between any two points. Then, define a particular distance, epsilon, that denotes the outer boundary of one point's neighborhood. Select any point as a starting point. All other points that are within that point's neighborhood are its neighbors. If the number of neighbors exceeds a predefined threshold, that point is a "core" point. All points in the neighborhood of a core point are in the same cluster. By then iteratively applying this algorithm to points in a core point's neighborhood, the cluster will expand until it reaches its edge. The algorithm's name perfectly describes its driving concept of tracking the density of points through space. Points in neighborhoods that aren't dense enough are marked as noise.

Distance Metrics

As alluded to above, a proper implementation of the DBSCAN algorithm allows a caller to specify one of a number of different distance metrics. Below, I briefly cover the

Manhattan/city block, Euclidean, Minkowski and cosine distance metrics. The Manhattan or city block distance metric gets its name from the observation that, when moving around in a city, the distance between any two points is very rarely the "as the crow flies" distance. When looking at a map of a North/South oriented grid-based city from a bird's eye view, the distance that it takes for someone walking that grid to travel on the streets from one point to another is the difference in the longitudinal distance added to the difference in latitudinal distance. In a Cartesian coordinate system, this is:

$$L_1 = |x_1 - x_2| + |y_1 - y_2|$$

This distance metric generalizes to higher dimensions *N*, where the distance between any two vectors *a* and *b* is:

$$L_1 = \sum_{k=1}^{N} |a_k - b_k|$$

This distance metric is often used when computing the distance a wire run in integrated circuits because they often only runs parallel to the X or Y axes (Black, 2006). It is also an appropriate distance metric when calculating the distance from the square a rook's currently occupies to any other square on a chessboard. Of the many distance metrics, this metric produces the greatest magnitude because it treats each dimension independently and then sums all of the magnitudes up. In comparison, the Euclidean distance, *D*, between two vectors is strictly not greater than the Manhattan distance. That is:

$$D_{euclidean} <= D_{manhattan}$$

The Euclidean distance between two points is the magnitude of the straight line between them. In a two-dimensional coordinate system, the Euclidean distance between two points *x* and *y* is:

$$L_2 = \sqrt{|x_1 - x_2|^2 + |y_1 - y_2|^2}$$

Generalized on vectors *a* and *b* in a space with *N* dimensions, it is:

$$L_2 = \sqrt{\sum_{k=1}^{N} |a_k - b_k|^2}$$

The Minkowski distance metric is a generalization of the Manhattan and Euclidean distance metrics. The Minkowski distance of order p between two vectors *a* and *b* in *N* dimensional space is:

$$L_p = (\sum_{k=1}^{N} |a_k - b_k|^p)^{1/p}$$

In other words, the Minkowski distance of the first order is another way of expressing the Manhattan distance and the Minkowski distance of the second order is an alias for the Euclidean distance.

Cosine similarity is often used in NLP. Given two vectors *a* and *b*, cosine similarity measures the size of the angle between the two vectors. If you were to envision a Cartesian plane, if the first vector is the x-axis and the second-vector is also on the x-axis, the angle between the two vectors would be *0* and *cos(0) = 1*. Thus a similarity of *1* means most similar. Two vectors that are orthogonal to each other–for example one that is on the x-axis and one that is on the y-axis–have a cosine similarity of *0*. Two vectors that are opposite of each other–for example one pointing towards *+x* and the other pointing towards *-x*–have a similarity of *-1*. More formally, cosine similarity between vectors *A* and *B* is:

$$\cos\theta = \frac{A \cdot B}{\|A\|_2 \|B\|_2}$$

For information retrieval tasks where the dimensions indicate term frequencies

there cannot be any dimensions of a vector who's values are negative, therefore cosine similarity in these cases is constrained to the range *[0, 1]*. Related to cosine similarity is a distance metric knows as cosine distance, which is denoted as:

$$cos_{distance} = 1 - cos_{similarity}$$

Any of the above distance metrics could be used with the DBSCAN algorithm. In my implementation I have chosen to use the second order Minkowski distance metric, as you'll see in Listing 9 for the DBSCAN algorithm. This is because my implementation already computes a unit vector using TF-IDF.

Term Frequency/Inverse Document Frequency

TF-IDF is a normalization procedure that can be applied to documents represented as vectors of terms. It comes from the observation that terms appearing more frequently indicate what a document's about, yet those terms appearing in many documents carry less unique meaning in any one document. For example, imagine our corpus has four documents as shown in Table 4.6:

Doc #	Words
1	the way that can be told
2	is not the true way
3	the names that can be named
4	are not the real names

Table 4.6: Corpus documents used to demonstrate TF-IDF

We can then create a vector with a dimensionality of the set of unique terms in the corpus. In the example above, there are twelve unique terms (after stemming). Each document can then be represented as a 12-dimensional vector, where the value in each dimension is the number of times that the particular term occurs in the document. A vector representation of each of the documents in shown in Table 4.7:

Term	Doc 1	Doc 2	Doc 3	Doc 4
the	1	1	1	1
way	1	1	0	0
that	1	0	1	0
can	1	0	1	0
be	1	0	1	0
told	1	0	0	0
is	0	1	0	1
not	0	1	0	1
true	0	1	0	0
name	0	0	1	1
be	1	0	1	0
real	0	0	0	1

Table 4.7: Term frequency vectors used in TF-IDF calculations

The term “real” is significant because it appears is only one document. If someone were to search for “real,” for example, the highest-ranked result would be Doc 4. On the other hand, the term “the” in the documents is meaningless because every document has it in exactly the same frequency. If C be the number of documents in the corpus, and C_x the number of documents in the corpus that contain a particular term, then the TF-IDF calculations is determined as follows:

$$TF(x) = \log_{10} x + 1$$

$$IDF(x) = \log_{10} C/C_x$$

Then, for every term in every document, replace its term count with v:

$$v = \frac{TF(x)}{IDF(x)} \text{ for } IDF(x) > 0$$

$$v = 0 \text{ for } IDF(x) = 0$$

These formulas encapsulate the fact that, (a) if a term is in every document it carries no meaning and, (b) that the number of occurrences of a term is a document increases that terms importance logarithmically, not linearly. Now that distance metrics and TF-IDF have been covered, you'll see applications of both concepts in the DBSCAN implementation in Listing 8 on page 65.

An implementation of the DBSCAN algorithm

The DBSCAN algorithm's steps are summarized in Figure 4.12:

1. Compute the distance between every sample.
2. For each sample, find all its neighbors.
3. Mark all samples whose neighborhood contains more than a minimum number of neighbors as "core."
4. Loop over all samples. Continue until finding a core point. If no more core points, exit.
5. Assign the core point to the current cluster. Perform a depth-first search on the samples in its neighborhood. Add all neighbors to a stack.
6. For each sample in the stack, determine if it is a core sample. If it is, go to step 4. If not, assign the sample to the current cluster.
7. When the stack is empty, go to step 3 and continue with the next sample.

Figure 4.12: An outline of the steps in the DBSCAN algorithm

This algorithm is implemented in Python starting with Listing 8:

```
def dbscan(X, eps, min_samples):
    """Runs DBSCAN with the given radius and number of samples
    in circle.

    Runs Density-Based Spatial Clustering of Applications with
    Noise, using a brute-force neighbors search algorithm.

    inputs:
        X - an array of shape (m, n) containing m samples and
        n features.

        eps - the radius within which to check for neighbors.

        min_samples - the number of samples that must be found
        within eps in order to consider a sample point to be a
        core point and within a cluster.

    outputs:
        is_core - an array of shape (m) that indicates whether
        each sample is a core in a cluster.

        labels - an array of shape (m) corresponding to X
        indicating what cluster a sample is in.  If a sample
        is found to be noise, its label is -1.
    """

    # construct m-by-m distance matrix
    d_mat = pdist(X)

    # find neighbors of each sample i
    neighbors = find_neighbors(d_mat, eps)

    # label the clusters using Depth-First Search
    is_core, labels = label_dbscan(neighbors, min_samples)

    return is_core, labels
```

Listing 8: The top-level DBSCAN function

The `pdist` function declared in the `distance` module is defined as:

```
def pdist(X, dist_fn=minkowski_dist, p=2):
    """Constructs a pairwise distance matrix.

    inputs:
        X - a standardized array of shape (m, n), with m samples
        and n features.

    outputs:
        an array of shape (m, m) with element X[i, j] filled with
        the distance between m[i] and m[j].
    """
    d_mat = np.zeros((X.shape[0], X.shape[0]))
    for i, j in product(range(d_mat.shape[0]), range(d_mat.shape[0])):
        d_mat[i][j] = dist_fn(X[i], X[j], p)

    return d_mat
```

Listing 9: Function for calculating the pairwise distance

And lastly, the generalized distance function is demonstrated in Listing 10:

```
def minkowski_dist(x, y, p):
    """Calculates the Minkowski distance between two points x and y

    The Minkowski distance is defined as sum(|x - y|^p)^(1/p).
    When p = 1, the Minkowski distance is the same as the Manhattan
    distance.  When p = 2, the Minkowski distance is the same as the
    Euclidean distance.

    Inputs:
        x, y - the two points to calculate the distance between.
        p - the p-norm

    Returns:
        The distance as a float
    """
    exp = p if p >= 0 else float(p)
    return np.sum(np.absolute(x - y) ** exp) ** (1 / exp)
```

Listing 10: Function for calculating the minkowski distance

As you can see, the `pdist` function creates an `m`-by-`m` redundant distance matrix. The matrix is considered redundant because the value at (m, n) is mirrored at (n, m). The `pdist` function defaults to using the Minkowski distance metric with `p=2`. This function

can be improved to use only half of the memory space using a flat, non-redundant, data structure. This is actually an option that SciPy's implementation of its `pdist` function provides. Another possibility is to use ball trees or kd-trees to determine the neighborhood of points instead of constructing distance matrices. This avoids the calculation of the full `m` by `m` distance matrix. The choice between calculating the full distance matrix or using a kd-tree should depend on the size of the input. Currently scikit-learn defaults to using kd-trees unless instructed otherwise (Pedregosa et al., 2011).

To improve performance, SciPy has also implemented the `pdist` function in native C code. Rather than crossing the interpreted/native boundary every time the distance function is executed, a pointer to a list of the samples is passed to the C code and a pointer to the filled condensed distance matrix is returned. Additionally, SciPy's C version takes advantage of parallel execution when possible, whereas the Python version in Listing 9 will not. Consequently, SciPy's C version is at least an order of magnitude faster than the pure Python version above.

Next, Listing 11 demonstrates how the neighbors of each point are found:

```
def find_neighbors(distance_matrix, eps):
    """Finds all neighbors within eps radius of each sample in the
    distance matrix.

    inputs:
        distance_matrix - a distance matrix of shape (m, m).  The
        upper-right triangle must be filled in, at a minimum.

        eps - epsilon, the value that two samples must be less than
        or equal to to be categorized as a neighbor.

    outputs:
        An array of shape (m), where each element i in the array is
        an array of the neighbors of X[i]
    """
    lst = [np.flatnonzero(row <= eps) for row in distance_matrix]
    return np.array(lst)
```

Listing 11: Function that finds the neighbors of a node

The `np.flatnonzero()` function returns the indices of all elements in a row of the

distance matrix that are closer than or equal to `eps`, which is the maximum distance a sample can be to still be considered a neighbor. Finally, Listing 12 clusters the points:

```
    def label_dbscan(neighbors, min_density):
        """Labels clusters and noise.

        inputs:
            neighbors - An array of shape (m), where each element i
            in the array is an array of the neighbors of X[i]

            min_density - The number of neighbors a sample must have
            to be considered a core sample.

        output:
            An array of size (m) of labeled clusters corresponding to
            which cluster neighbors[i] is in.  If neighbors[i] is
            noise, the output for that row is set to -1.
            """

        # initialize is_core array to m[i]'s neighbors >= min_samples
        # and labels array to -1
        is_core = np.array([len(el) >= min_density for el in neighbors]) #1
        labels = -np.ones_like(is_core, dtype=int)                        #2

        lifo = []                                                         #3
        label = 0

        for i in range(len(neighbors)):                                   #4

            if labels[i] != -1 or not is_core[i]:
                continue

            while True:

                if labels[i] == -1:
                    labels[i] = label                                     #5
                    if is_core[i]:
                        lifo.extend(neighbor for neighbor in neighbors[i]
                                    if labels[neighbor] == -1)            #6

                if len(lifo) == 0:                                        #7
                    break
                i = lifo.pop()                                            #8

            label += 1

        return is_core, labels
```

Listing 12: Function that tags each sample as belonging to a cluster

The numbers in the comments to the right of the code in Listing 12 are further explained in Figure 4.13:

1. An boolean array is initialized. The length of the array is the same as the number of samples being clustered. If sample `i` has at least `min_density` neighbors, then index `i` in the array is True.
2. An int array the same size as the `is_core` array is initialized with -1s, indicating that no samples have yet been assigned to a cluster.
3. A LIFO is used in the depth-first search that identifies all samples in a radius with at least a given density.
4. Each sample is visited at least once.
5. If the sample is a core sample but hasn't yet been given a cluster label, then use the current cluster label for that new cluster.
6. Add all of the core cluster's neighbors to the LIFO.
7. If the LIFO is empty, jump to #4 and process the next sample.
8. Process the items in the LIFO until there are none left.

Figure 4.13: Explanation of the `find_neighbors` function

To demonstrate how this function performs, I implemented a demo that clusters a few hundred samples. Each sample has two features and thus can be graphed onto the Cartesian plane. Euclidean distance is used as the distance metric. The demo demonstrates how varying the two inputs to the `dbscan` function, `eps` (radius) and `min_samples`, affects the clustering. In Figure 4.14, `eps` is incrementally increased from 0.2 to 0.65 while `min_samples` remains constant at 25. Noise points are denoted by being black. As `eps` grows, DBSCAN goes from not identifying any clusters to annotating all of

the points as belonging to a single cluster. When `eps` is 0.3, the algorithm identifies three separate clusters while still leaving much of the samples on the boundaries as noise.

If, on the other hand, `eps` remains constant at 0.3 and the number of samples within a 0.3 radius needed to constitute a cluster (`min_samples`) is varied, the opposite effect occurs. Large, inclusive clusters are slowly reduced to small, dense clusters, to the point that the algorithm doesn't identify any clusters at all. This is shown in Figure 4.15.

`eps` and `min_samples` are really two sides of the same coin, as together they determine density. A refinement of the DBSCAN algorithm as implemented in Listing 8 on page 65 would normalize the input data and ask for a single input, `density`.

4.4.3 The Python STL's LRU Cache Algorithm

A cache is used as a proxy for some computation to prevent an expensive operation from occurring if that expensive operation has already been performed in the past. By using the cache as a proxy, the cache might be able to return the result directly if it had stored the result of the previous computation, thereby bypassing the slow operation. An LRU cache is a cache that is optimized for both time and space. It maintains a list of its elements ordered by last-accessed time and, when full, ejects elements that have been last-accessed the longest time ago. This cache replacement policy is best applied with access patterns that exhibit locality of time and space, *i.e.*, where the statement that elements or addresses accessed most recently are those most likely to be accessed again soon is true.

There are many alternative cache replacement policies. For example, there's the most-frequently-used (MFU) cache replacement policy, which orders its element by access time the same way that an LRU cache does, but rather than ejecting an element that was used the furthest in the past it ejects the element that was last accessed. While counter-intuitive, according to Chou and DeWitt (1986) in certain situations the MFU cache replacement policy performs pretty well.

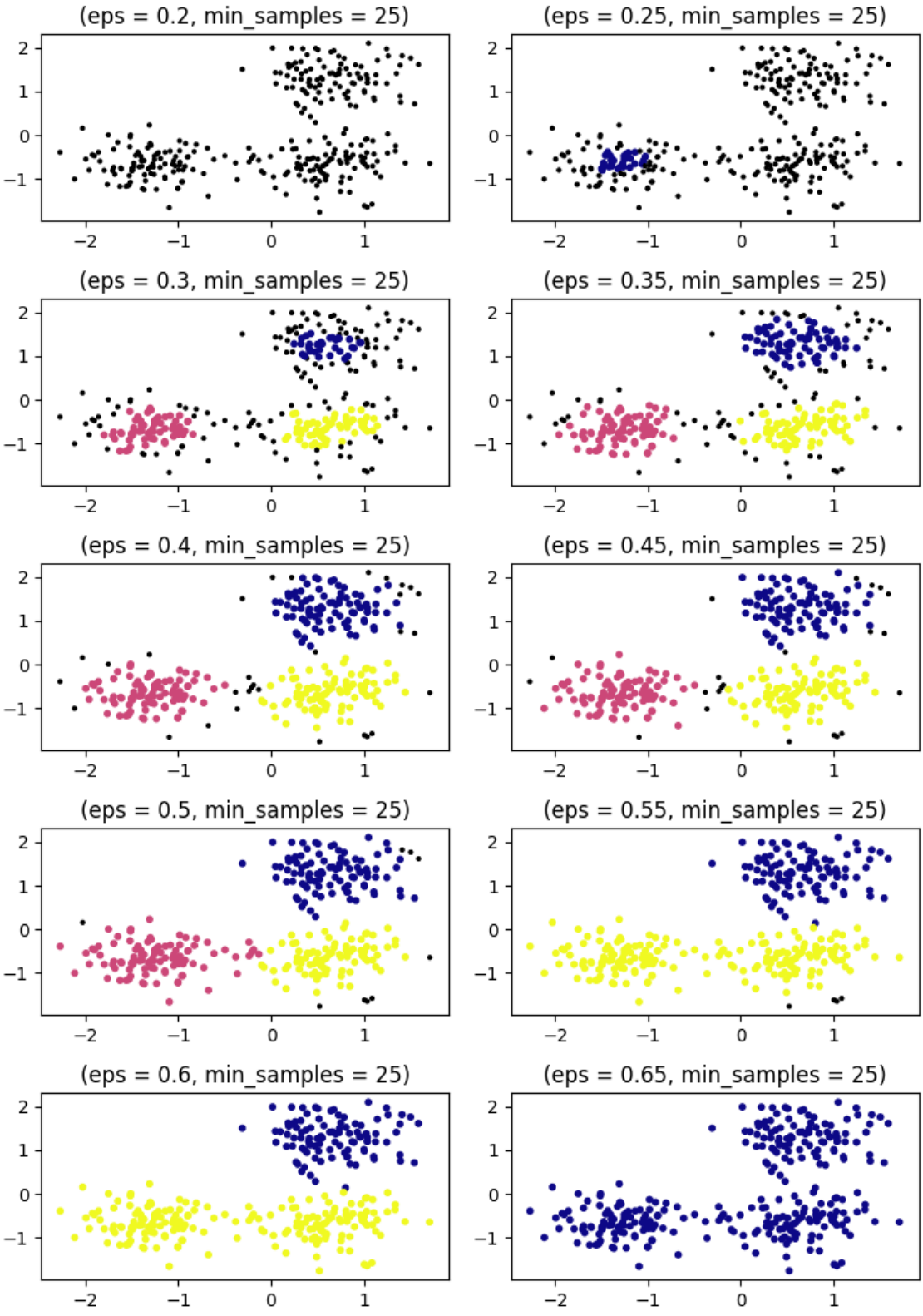

Figure 4.14: Varying the eps parameter of Density-Based Spatial Clustering of Applications with Noise

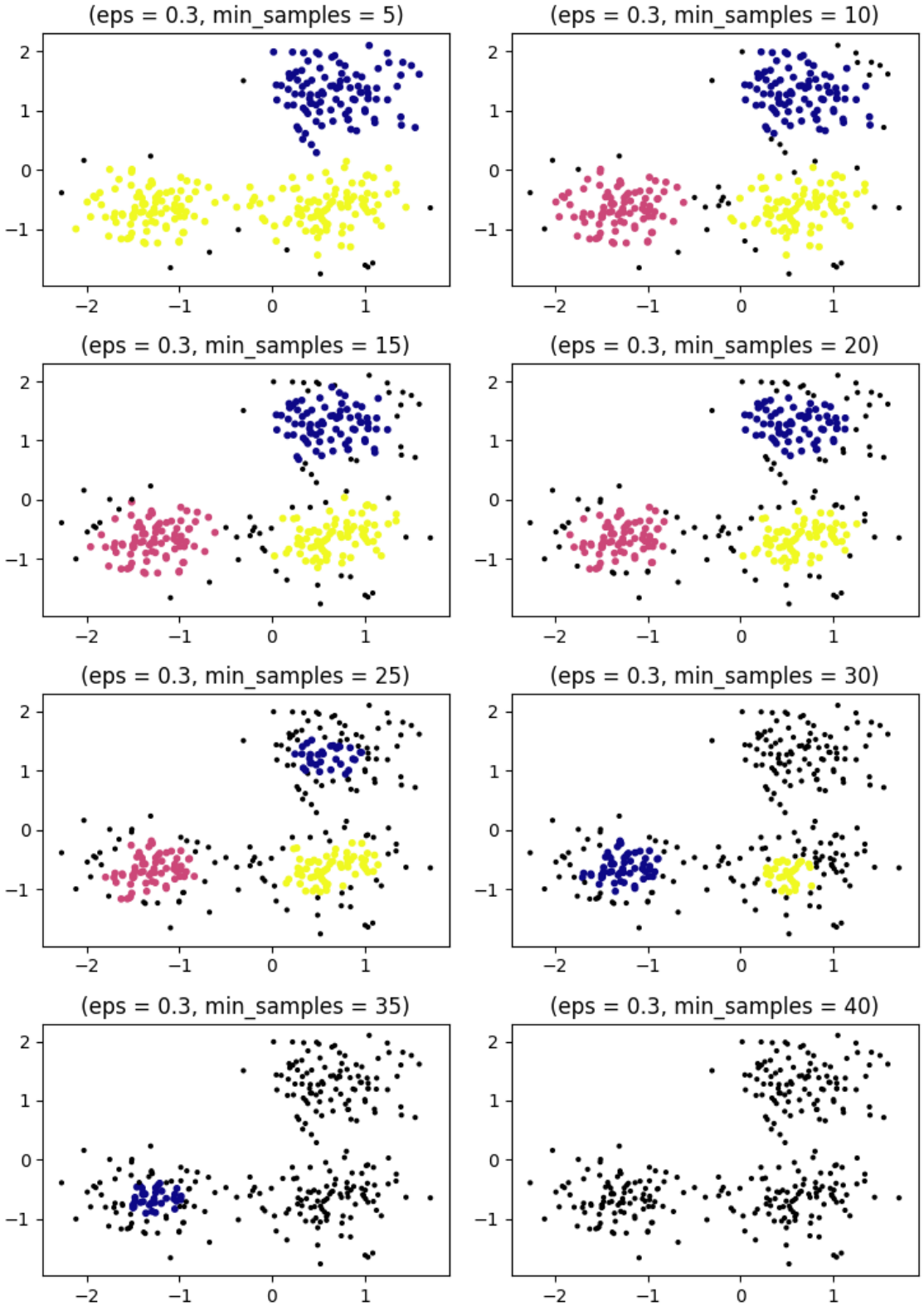

Figure 4.15: Varying the minimum number neighbors constituting a cluster in Density-Based Spatial Clustering of Applications with Noise

Another alternative to the standard LRU is a time-based LRU, which adds a timestamp as an input to the ejection-selection function. More concretely, an item in the cache has a time-to-live (TTL). When an element's TTL is exceeded, the entry is no longer be considered valid in the cache and may either be ejected or recomputed. There's also the least-frequently-used (LFU) replacement policy, which counts the number of times an element in the cache has been accessed and replaces the element that has been accessed the least number of times.

The article service uses Google Translate to translate Chinese and English text into different languages. Google Translate's pricing is based on the number of characters translated, so the cost function to minimize is the number of characters translated. Making the call over the public internet is also slow, and Google rate-limits the number of characters that can be translated per minute. I ran into both of these issues while developing the article service.

Intuitively, one expects that a word used in a news article is likely to be repeated later in that news article and perhaps even within other news articles from the same date. As time passes, the topics covered in news articles change and words associated with old topics no longer serve much of a purpose in a cache. Under these assumptions, a LRU cache replacement policy for translations from the Google Translate API is an appropriate application of a LRU cache. Luckily, the STL supplies a LRU cache implementation. What follows is an explanation of the LRU algorithm and how it is implemented in the STL.

An implementation of an LRU cache

A software-defined LRU algorithm needs a mapping data structure, such as a hash-map, and a circular doubly-linked list. The mapping data structure holds the elements currently in the cache, and the doubly- linked list contains the bookkeeping of the order that elements in the cache have last been accessed. Each node in the

doubly-linked list has three members: a link to the previous node, a link to the next node, and a key to the mapping, which can be empty. Only one node in the doubly-linked list can have its key member empty, because an empty key indicates the root of the doubly-linked list. The doubly-linked list is operated as a queue. Conceptually, elements at the rear of the queue are ejected when the queue is full and new elements are added to the front of the queue. Elements accessed when they are in the cache are moved to the front of the queue. The root element provides access to the front and rear of the queue through its previous and next links. Rather than allocating a new linked-list node when an element not in the cache is accessed and freeing the old nodes, an optimization taken by the STL is to rotate the root around the circular linked-list to the ejected element and reuse the old root's node. In such a way the doubly-linked list, once fully-allocated, never allocates or frees any additional memory.

To demonstrate the usage of the bookkeeping circular doubly-linked list, below is a example run of a LRU cache with four spaces and an access pattern of 1, 2, 3, 4, 3, 5, 1.

Figure 4.16 shows the first step, where the linked list is first initialized. The root element's key is empty (0) and its previous and next links point to itself.

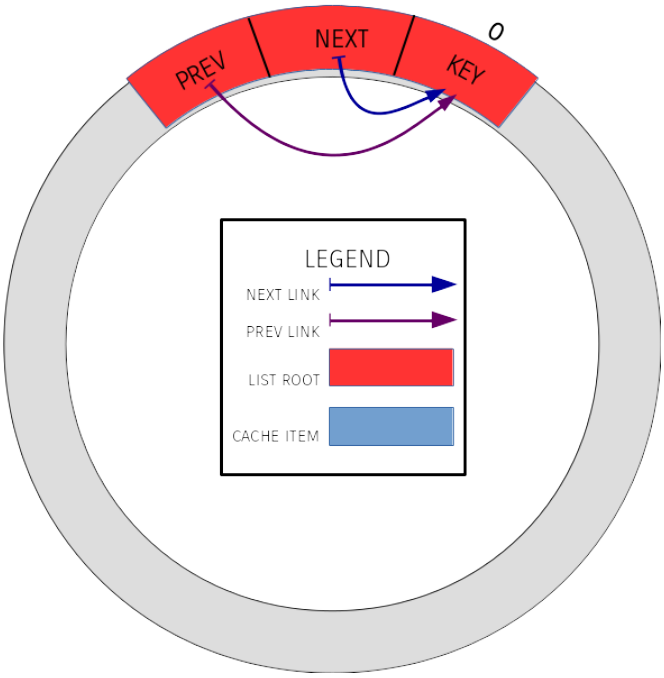

Figure 4.16: Initializing a doubly-linked list for bookkeeping in the LRU cache. *Note the legend, subsequent figures omit this legend to save space.*

When element 1 is accessed there's a check to determine whether the cache is full and, since it's not, the new element is added into the linked list. Its next and previous links point to the root, and the root's next and previous links point to it. The "next" link from the root is the element in the list that has been most-recently accessed, and the "previous" link is the element that has been least-recently accessed and will be replaced when space in the cache is needed.

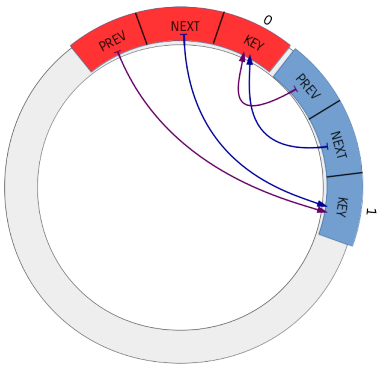

Figure 4.17: Element 1 is accessed

The same thing happens as elements 2, 3, and 4 are accessed:

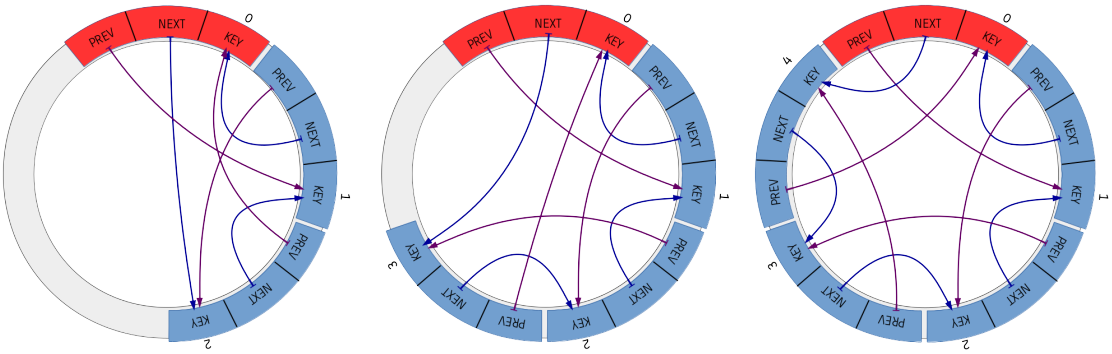

Figure 4.18: Elements 2, 3, and 4 are accessed

At this point, element 3 is accessed again. Since the element is already in the cache, its cached value is returned. The links in the linked list are updated so that the root's next element is now element 3 and its previous link still points to element 1.

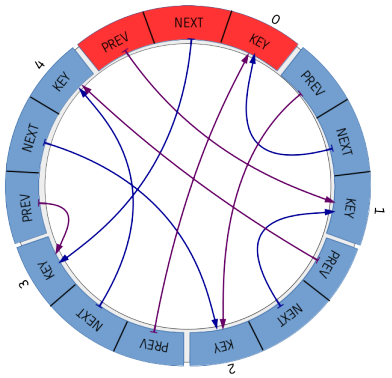

Figure 4.19: Element 3 is accessed again

When element 5 is accessed it's not in the cache and there's no room in the cache to add new elements. Consequently, the least-recently-used element will be ejected. Since the algorithm uses a circular doubly-linked list, this can be achieved by simply moving the root to its "previous" link and replacing the space where the root was with the element that was just accessed—in this case, element 5.

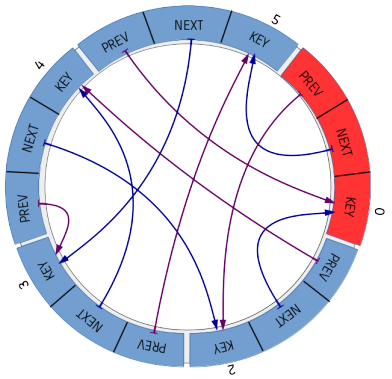

Figure 4.20: Element 5 is accessed and replaces the root

Finally, element 1 is accessed. Again, it's not in the cache so the root is pushed forward one space and the old root's key is set to element 1.

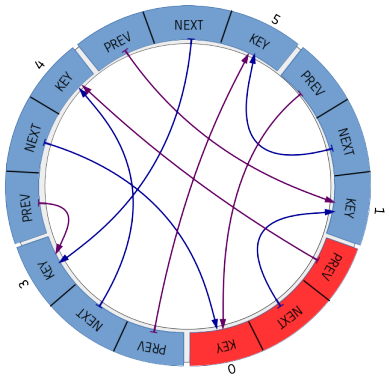

Figure 4.21: Element 1 is accessed and replaces the root

This particular approach to a software-defined LRU cache was implemented first in the STL with Python release 3.2, published on February 20, 2011, as a pure-Python decorator in the `functools` module. It was then subsequently improved upon in Python 3.3 and a faster C implementation was included in the 3.5 release, which Storchaka (2012) reported achieved an execution speedup on the order of 25x. Both the pure Python and C implementations used the same circular doubly-linked list approach shown above. Since caching is a generic concept that could be applied to many different use cases, the developers who wrote it chose to implement the `lru_cache` as a Python decorator, giving it the ability to decorate any particular function.

Python decorators are syntactic sugar that are particularly helpful for certain use cases. Not to be confused with the object-oriented decorator design pattern, which decorates or wraps an object at runtime with additional or overridden behavior, Python decorators serve the purpose of adding cross-cutting functionality to a function or class at the point where it is defined. Pragmatically, a decorator wraps a function with code that will be executed before and/or after the function is called. The `lru_cache` is a great example of a decorator that can be applied to many different functions. It is easy to imagine needing a cache for database queries, HTTP API calls, or a recursive

implementation of the Fibonacci sequence. Further information on decorators and the `functools.lru_cache` decorator's pure Python implementation can be seen in "Glossary" (Python Software Foundation, 2017) and the source code in the STL's `functools.py` (Hettinger et al., 2017).

Chapter 5

Results and Observations

The best clusters are made from nouns

A few surprising results arose through the process of designing and implementing this cross-language article clustering application. First, determining similarity between news articles based on their nouns alone provided the best clusters. This was not what I expected. I had expected that, given that one word in the source language can often be translated into many words in the target language, the probability that it would be translated to the corresponding word used in the target language would be low. For example, I anticipated that the Chinese word for car (汽车) might be translated as auto or automobile and thus not match English articles that used the word car. I reasoned that to account for these discrepancies, the best thing to do would be to expand every word in each article by all other words that carry the same meaning (their synsets). When put into practice, synset expansion noticeably degraded clustering performance and actually created subjectively worse clusters.

There are two possible reasons why synset expansion did not improve article clustering. The first is that the translation system employed, Google Translate, provides the most-probable translation for a particular word. A part of determining which translation is most probable is how frequently the proposed translation is used in the target

language. While the most-probable translation of a word in some cases may not match the word used in English articles about the same topic, on average it chooses the right translation frequently enough. Words rarely appear alone in a document, which is one of the key insights driving LSA, so while one word may be translated incorrectly for a particular context, it's likely that the words around it won't be.

A second possible reason why synset expansion is unnecessary is that nouns are less ambiguous than other parts of speech in Chinese. This statement must be qualified because the parts of speech themselves in Chinese are themselves often ambiguous. In Chinese, the same word can be used as a noun or a verb. This is not completely foreign to English (We google things on Google), but it's a core feature of the Chinese language. Kwong and Tsou (2003) provides the example of 怀疑, which can be an adjective (suspicious), a verb (to suspect), or a noun (suspicion). Yet, if the part of speech is identified correctly, the space of possible translations is made much smaller. Thus, much of the burden of winnowing the search space of translations is pushed onto Stanford CoreNLP, which is responsible for tagging each word an article with a part-of-speech. Stanford CoreNLP seems to do a good job of tagging nouns in Chinese. Unfortunately, Google Translate's API does not allow one to specify which part of speech a translation should fulfill, but when a word has multiple translations, Google Translate usually ranks the noun first. This raises the possibility of using more appropriate machine translation system that allows one to provide part-of-speech context to the software as input and falls back to Google Translate for cases such as new words or named entities where Google Translate performs remarkably well.

Most articles are distant in vector space

It was surprising to find both in hierarchical and density-based clustering algorithms that the Euclidean distance between two even seemingly similar articles is large. This is shown most clearly in Figure 5.1, a sample hierarchical clustering run:

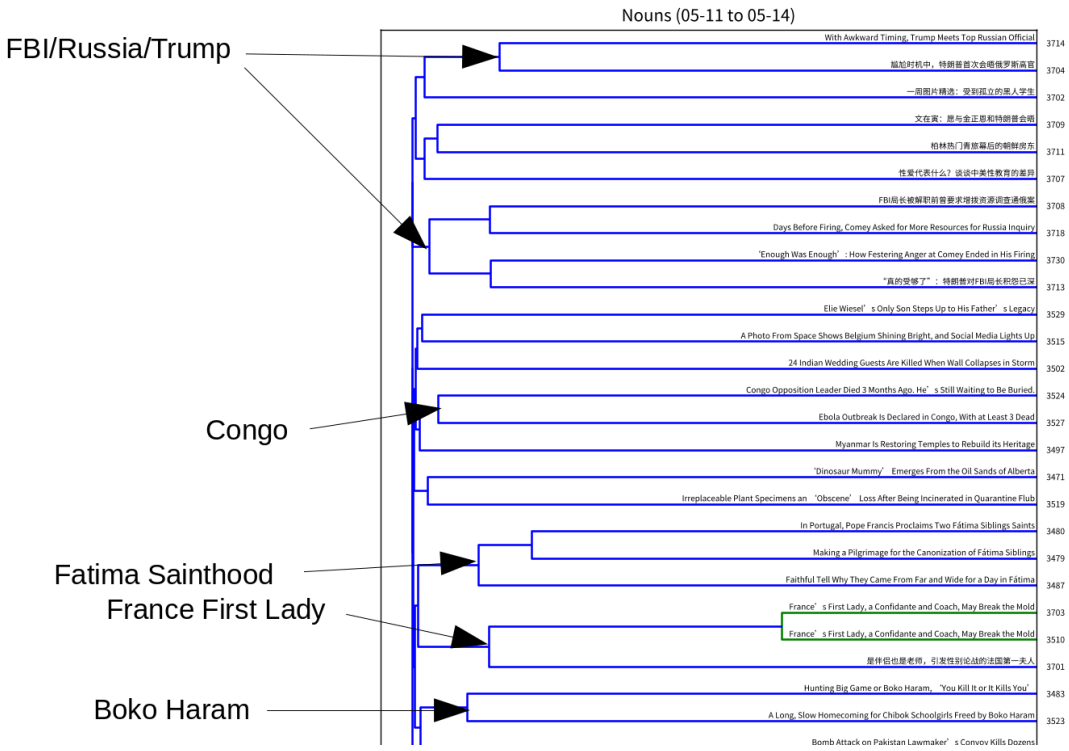

Figure 5.1: Part of an example clustering from May 11-15, 2017. The unit-normalized vectors are very distant from each other.

Figure 5.1 demonstrates that even articles written in the same language and that share many, but not all, of the same paragraphs (“France’s First Lady, a Confidante and Coach, May Break the Mold”) have a normalized distance of around 0.6 from each other. Articles that are also very similar topically, such as the three on Fatima’s Sainthood, are around 1.2 units from each other. This was an unexpected result, yet upon further reflection it makes sense that the higher the dimensionality of the vector space, the easier it is for two vectors to be further away from one another. This phenomenon occurred when using the DBSCAN algorithm also, and through iterative fine-tuning I found that a good `eps` to set as the intra-neighborhood distance for the algorithm is 1.3 units.

Performance bottlenecks can come from unexpected places

When developing the clustering algorithms, I knew that many of them are computationally expensive in terms of space and time. Consequently, my implementations of the clustering algorithms use numpy array-based operations where possible. This means that rather than operating on individual elements in an array, as is typically in a for-loop in a programming language, numpy operates on entire rows or even matrices at once. Many of Numpy's array-based operations are written in in a way using native C extensions that optimizes parallel execution. This undoubtedly led to faster execution of the algorithms, but my initial performance concerns remained with the clustering algorithms.

Yet unexpectedly, one of the main culprits elongating the execution time of the web application turned out not to be the clustering algorithm itself but the retrieval of the CoreNLP JSON objects stored in the database. There are two reasons why storing the CoreNLP objects as JSON in the database negatively impacts performance: (a) each object is large and, (b) each object must be deserialized into a Python dictionary before it can be operated on. The average size of an object in the `corenlp_json` column of the `articles` table is 187KB. If there are 100 articles to cluster for a given timespan, then approximately 20MB of data must be transferred to the Django web app and deserialized before the clustering can begin. In most scenarios where the database and web application were separated by a network this amount of data transfer would prove untenable.

In the case of the web app, the story card page that users first see when visiting the site performs a total of 11 queries. In Figure 5.2, the top row of the query profiling page in the Django Debug Toolbar shows the single query that requests the actual JSON objects takes over half of the total query time:

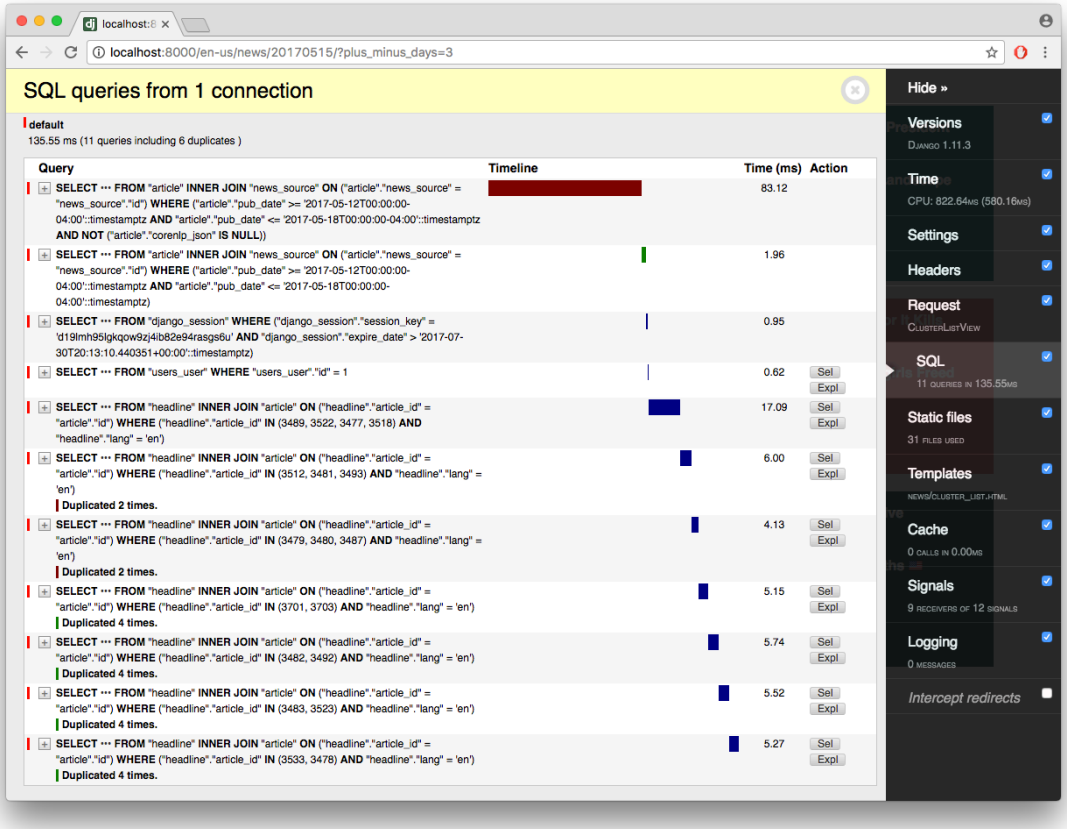

Figure 5.2: The query retrieving the `corenlp_json` column is by-and-far the most expensive computationally

This is just for the transfer of the data to the Django application; the deserialization of the JSON adds additional time. While spending 100s of milliseconds in the current system is concerning, it is not yet a blocking performance issue. As additional news sources are added and the number of articles to be clustered increases, it will undoubtedly become one. This unexpected performance bottleneck will be taken care of in a future iteration of the web application by modifying the database schema and token retrieval algorithm. This change would entail creating a table for each of the different features of an article, such as nouns and verbs, and populating them by the article service. Then, when the web application requested all of the nouns, it would simply request all of the elements of a table whose `article_id` corresponded to an article within a date range and return

rows of words. This will require much less data transfer and no JSON deserialization.

Chapter 6

Summary and Conclusions

To the best of my knowledge, the application developed in this thesis is currently the only application on the internet today that clusters English and Chinese news articles. This thesis's impetus is the idea that it's important for Americans to understand Chinese perspectives and Chinese to understand American perspectives. It described how the news media is one of the main arteries of national bias and hypothesized that providing a platform that allows Americans to read Chinese news articles and vice versa can be a tool to combat that bias. To that end, it explored how best to create an application that topically clusters news articles across languages. It found that NLP techniques, such as part-of-speech tagging, assisted greatly to identify which words from a news article should be used as input to an unsupervised clustering algorithm. Experimentation found that density-based clustering worked very well for news articles, as algorithms like DBSCAN account for noise and there is a lot of noise in news topics. Hierarchical clustering algorithms were explored but presented obstacles that made them less conducive to topical clustering. An implementation of the web application portion of the thesis can be viewed on the author's website.[1] The source code repository can be viewed online as well.[2]

If I were to start the thesis today, I would be very interested in looking into how

[1] `http://www.plaintexttransmissions.org/news`
[2] `http://www.bitbucket.org/gusennan/clds-web`

LSA could be used in lieu of document translation to determine document similarity. The different language versions of Wikipedia with their cross-language links are rich knowledgebases to mine. I would also design the database schema differently to avoid the performance issue associated with retrieving large JSON objects, as noted in Chapter 5. Lastly, the application is lacking objective measurements of its clustering quality. If I were to start again today, I would first create a corpus of news articles in English and Chinese and manually create clusters, thereby having a gold-standard to compare the output of different clustering algorithms to. I would also add a feedback mechanism into the web application that allows users to provide real-time feedback on the quality of clusters.

There is ample future work that this thesis could serve as a starting point for. First, it would be very interesting to conduct user studies to determine whether or not using this application promotes cross-cultural understanding. Second, the techniques used in this thesis are by no means limited to the English and Chinese languages, future work could add additional languages to the application. Third, as noted above, it would be particularly interesting to devise a document similarity metric absent foreign language translation. As the source code for this thesis is available online, if anyone is so inspired to pursue this path of inquiry, the process of replacing the functions that determine document similarity should not be too difficult to implement.

References

Bird, S., Loper, E., & Klein, E. (2009). *Natural language processing with python*. O'Reilly Media Inc.

Black, P. E. (2006). Manhattan distance. In V. Pieterse & P. E. Black (Eds.), *Dictionary of Algorithms and Data Structures*. Retrieved from https://xlinux.nist.gov/dads/HTML/manhattanDistance.html

Brown, R. (2015). Django vs Flask vs Pyramid: Choosing a Python web framework. [Web log comment]. Retrieved from https://www.airpair.com/python/posts/django-flask-pyramid

Chou, H.-T. & DeWitt, D. J. (1986). An evaluation of buffer management strategies for relational database systems. *Algorithmica, 1*(1-4), 311–336.

Django Debug Toolbar developers and contributors. (2017). Django Debug Toolbar. (Version 1.7) [Software]. Available from https://github.com/jazzband/django-debug-toolbar.

Django Software Foundation. (2017). Django. (Version 1.11.3) [Software]. Available from https://djangoproject.com.

Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd* (Vol. 96, *34*, pp. 226–231).

Evans, D. (2005). *Identifying similarity in text: Multi-lingual analysis for summarization* (Doctoral dissertation, Columbia University). Retrieved from

http://search.proquest.com.ezp-prod1.hul.harvard.edu/docview/305007245?accountid=11311

Evans, D., Klavans, J. L., & McKeown, K. R. (2004). Columbia newsblaster: Multilingual news summarization on the web. In *Demonstration Papers at HLT-NAACL 2004* (pp. 1–4). HLT-NAACL–Demonstrations ’04. Boston, Massachusetts: Association for Computational Linguistics. Retrieved from http://dl.acm.org/citation.cfm?id=1614025.1614026

Fingar, T. & Fan, J. (2013). Ties that bind: Strategic stability in the U.S.-China relationship. *The Washington Quarterly*, *36:4*, 125–138. doi:10.1080/0163660X.2013.861718

Foreign Ministry of the People’s Republic of China. (2017, April 8). Introduction to the Mar-a-Lago summit of the US and Chinese heads of state. Retrieved from http://www.fmprc.gov.cn/web/ziliao_674904/zt_674979/dnzt_674981/xzxzt/xjpdfljxgsfw_689445/zxxx_689447/t1452260.shtml

Fuxjäger, M. (2017). Postgresql. Retrieved August 20, 2017 from the Python Wiki: https://wiki.python.org/moin/PostgreSQL.

Greenfeld, D. R. (2017). Cookiecutter Django. (Version 1.5.1) [Software]. Available from https://github.com/pydanny/cookiecutter-django.

Greenfeld, D. R. & Greenfeld, A. R. (2015). *Two scoops of Django: Best practices for Django 1.8* (3rd ed.). Los Angeles, California: Two Scoops Press.

Gregorio, F. D. & Varrazzo, D. (2016). *Frequently asked questions*. Retrieved from http://initd.org/psycopg/docs/faq.html

Hassan, S. & Mihalcea, R. (2009). Cross-lingual semantic relatedness using encyclopedic knowledge. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3* (pp. 1192–1201). EMNLP ’09. Singapore: Association for Computational Linguistics. Retrieved from http://dl.acm.org/citation.cfm?id=1699648.1699665

Hees, J. (2015). *Scipy hierarchical clustering and dendrogram tutorial*. Retrieved from https://joernhees.de/blog/2015/08/26/scipy-hierarchical-clustering-and-dendrogram-tutorial/

Hettinger, R., Storchaka, S., Coghlan, N., Peterson, B., Wouters, T., Brandl, G., … Belopolsky, A. (2017). functools.py. [Online Git Repository]. Retrieved from https://github.com/python/cpython/blob/6f0eb93183519024cb360162bdd81b9faec97ba6/Lib/functools.py

Hipp, D. R. (2008). *Appropriate uses for SQLite*. Retrieved from https://sqlite.org/whentouse.html

Hoffman, P., Graña, D., Olveyra, M., García, G., Cetrulo, M., Bogomyagkov, A., … Ramírez, N. (2017). Scrapy. (Version 1.3.3) [Software]. Available from https://scrapy.org/.

Hunter, J. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. doi:10.1109/MCSE.2007.55

Jones, E., Oliphant, T., Peterson, P., et al. (2001). Scipy: Open source scientific tools for Python. Retrieved from http://www.scipy.org

Kwong, O. Y. & Tsou, B. K. (2003). A synchronous corpus-based study of verb-noun fluidity in Chinese. In *Proceedings of the 17th Pacific Asia Conference on Language, Information and Computation (PACLIC 17)* (pp. 194–203).

Lemburg, M.-A. (1999). *Python database API specification v2.0* (Python Enhancement Proposals No. 249). Retrieved from https://www.python.org/dev/peps/pep-0249/

Lönnberg, M. & Yregård, L. (2013). *Large scale news article clustering* (Master's thesis, Chalmers University of Technology, Gothenberg, Sweden).

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., & McClosky, D. (2014). The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations* (pp. 55–60). Retrieved from http://www.aclweb.org/anthology/P/P14/P14-5010

Martin, T. (2017, May 23). Dissecting trump's most rabid online following. *FiveThirtyEight*. Retrieved from https://fivethirtyeight.com/features/dissecting-trumps-most-rabid-online-following/

McInne, L. (2017). *Building and exploring a map of Reddit with Python*. Retrieved from https://lmcinnes.github.io/subreddit_mapping/

McKeown, K. R., Barzilay, R., Evans, D., Hatzivassiloglou, V., Klavans, J. L., Nenkova, A., … Sigelman, S. (2002). Tracking and summarizing news on a daily basis with Columbia's newsblaster. In *Proceedings of the Second International Conference on Human Language Technology Research* (pp. 280–285). HLT '02. San Diego, California: Morgan Kaufmann Publishers Inc. Retrieved from http://dl.acm.org/citation.cfm?id=1289189.1289212

McKeown, K., Barzilay, R., Chen, J., Elson, D., Evans, D., Klavans, J., … Sigelman, S. (2003). Columbia's newsblaster: New features and future directions. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology: Demonstrations - Volume 4* (pp. 15–16). NAACL-Demonstrations '03. Edmonton, Canada: Association for Computational Linguistics. doi:10.3115/1073427.1073435

McKeown, K., Passonneau, R. J., Elson, D. K., Nenkova, A., & Hirschberg, J. (2005). Do summaries help? In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 210–217). SIGIR '05. Salvador, Brazil: ACM. doi:10.1145/1076034.1076072

McKinney, W. (2010). Data structures for statistical computing in python. In S. van der Walt & J. Millman (Eds.), *Proceedings of the 9th python in science conference* (pp. 51–56).

Mitchell, R. (2015). *Web scraping with python: Collecting data from the modern web*. O'Reilly Media, Inc.

Munson, S. A., Lee, S. Y., & Resnick, P. (2013). Encouraging reading of diverse political viewpoints with a browser widget. In *ICWSM*.

Park, S., Kang, S., Chung, S., & Song, J. (2009). Newscube: Delivering multiple aspects of news to mitigate media bias. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 443–452). CHI '09. Boston, MA, USA: ACM. doi:10.1145/1518701.1518772

Pazzanese, C. (2017, June 1). The troubling U.S.-China face-off. *Harvard Gazette*. Retrieved from http://news.harvard.edu/gazette/story/2017/06/new-book-evaluates-the-u-s-china-face-off/

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Python Software Foundation. (2017). Glossary. In *Python 3.6.2 Documentation*. Retrieved from https://docs.python.org/3/glossary.html

Steinberger, R., Pouliquen, B., & Hagman, J. (2002). Cross-lingual document similarity calculation using the multilingual thesaurus EUROVOC. *Computational Linguistics and Intelligent Text Processing*, 101–121.

Storchaka, S. (2012). C implementation of functools.lru_cache. [Msg 177953]. Message posted to https://bugs.python.org/msg177953.

The PostgreSQL Global Development Group. (2017). PostgreSQL. (Version 9.6.3) [Software]. Available from https://www.postgresql.org/downloads.

Walt, S. v. d., Colbert, S. C., & Varoquaux, G. (2011). The numpy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, *13*(2), 22–30.

Woese, C. R., Kandler, O., & Wheelis, M. L. (1990). Towards a natural system of organisms: Proposal for the domains archaea, bacteria, and eucarya. *Proceedings of the National Academy of Sciences*, *87*(12), 4576–4579.

Glossary

API application programming interface.

AWS Amazon Web Services.

cron A software program, usually found on computers with Unix-like operating systems, that allows one to schedule tasks to run on predetermined intervals. These tasks are usually referred to as cron jobs.

DBSCAN Density-Based Spatial Clustering of Applications with Noise.

dendrogram A tree diagram, where each node in the tree has at most two children. Frequently used to describe evolutionary relationships between species and results of hierarchical clusterig algorithms.

Django Debug Toolbar A debug tool one can include in their Django-based website to provide deeper insights into various aspects of Django's processing of HTTP requests and responses .

ESA explicit semantic analysis.

EU European Union.

GIL global interpreter lock.

Gunicorn Stands for "Green Unicorn". A WSGI-compliant Python HTTP server.

HAC Hierarchical Agglomerative Clustering.

HDBSCAN Hierarchical Density-Based Spatial Clustering of Applications with Noise.

LFU least-frequently-used.

LRU least-recently used.

LSA Latent Semantic Analysis.

MFU most-frequently-used.

Nginx An open-source HTTP and reverse-proxy server.

NLP Natural Language Processing.

NLTK Natural Language Toolkit.

ORM object-relational mapper.

PostgreSQL An open-source relational database.

SAAS software-as-a-service.

selective exposure the process of filtering out messages that do no match our beliefs because of political preference, or simply by selecting a certain information medium.

Stanford CoreNLP An open-source toolkit maintained primarily by Stanford University for extracting information from human languages.

STL Python Standard Library.

subreddit A discussion topic on the website Reddit. A subreddit can be open for public viewing or comments or or limited to a particular restricted set of users.

synset a group of semantically equivalent words.

TF-IDF term frequency/inverse document frequency.

TTL time-to-live.

VM Virtual Machine.

WSGI Web Server Gateway Interface.