Opinion Mining & Summarization Final Report

Ernesto Cortes Groman, Kipp Dunn, Sar Gregorczyk, and Alex Schmidt

CS4624: Multimedia, Hypertext, and Information Access
Instructor: Dr. Edward Fox

May 2, 2018, Virginia Tech, Blacksburg VA 24061

Table Of Contents

Table Of Figures

Abstract

Opinion Mining Summarization is a Multimedia, Hypertext, and Information Access capstone project proposed by Xuan Zhang to Dr. Edward Fox. The purpose of the project is to generate a suite of tools that can generate useful data about products sold on the internet. The final goal is a suite of tools that, when given a product, can scrape the web for review data on that product and create easily accessible summaries of these reviews. This will allow the user to see the general opinion of online consumers on a given product.

In our design phase, we divided the overall project into four main deliverables: a web-crawler, database, web application, and a suite of summarization tools (see inventory for more details). To begin our development process, we identified open source libraries that performed some of the functionality our tools would need. From these libraries, we were able to begin developing tools specific to the needs we identified during our research phase.

We practiced test-driven development, frequently testing our tools on example websites and sample data, in order to ensure correctness and identify any needed design changes. For example, as the project progressed, simulated user testing identified the need for a more user-friendly way to interact with the tools. This led us to design a web application to provide a GUI for the program. Through this web application, it was planned that the user would be able to generate and browse product review summarizations, as well as start web-crawling requests in real time.

At the conclusion of the project, we have a full, cohesive tool. Through the web application, the web-crawler Python scripts can be used, review and summarization data is stored in a MySQL database, a variety of Python summarization scripts can be run on review sets, and the results can be cleanly viewed.

Throughout the process of this project, we learned a great deal about full-stack development. Everything we interacted with provided us with a new opportunity for learning and growth, whether it was Python scripting or the .NET framework. As well, integrating multiple tools written in different languages provided a new challenge for our team, beyond what we had experienced in previous classes. Overall, the start-to-finish completion of a major project was an excellent learning experience that will serve us well as we approach graduation and our future careers.

Introduction

Proposed by Xuan Zhang to Dr. Edward Fox as a project for the Multimedia/Hypertext/Information Access capstone, this project is related to Machine Learning, Natural Language Processing, and Information Retrieval.

The project aims at mining useful information, such as opinion summaries, quality problems, and solutions -- from user reviews of products or services. This is accomplished through the use of web-crawlers to scrape review data for specified products from online forums and stores. For each review, the web-crawler should locate and store the date, rating, title, text, and helpfulness (if present) of each review.

This data can then be stored into our database for easy access by the summarization tools. These tools generate opinion summaries for the review data using selective (e.g., TextRank), model-based (i.e., Latent Dirichlet Allocation), and potentially abstractive (e.g., Sequence-to-Sequence neural networks) methods. The summaries will be made available through a web application.

All of these tasks are combined in a seamless user experience using a web application. With this application, a user can search for a new product which will result in the summarization, storage, and web application reload with the new product available. This provides an easy tool with which to gather information about product opinions on the internet.

Requirements

This project can clearly be divided into four parts: 1) a web-crawler to collect large amounts of review data; 2) a database to store the collected data; 3) a suite of tools to summarize the stored data which then stores the summaries back into the database; and 4) a web application for interfacing with all the parts and providing an interface between the program and the user.

Web-crawler

In order to have review data to summarize, that data first has to be collected from the internet. This is the purpose of the web-crawler. It is a required deliverable given in the project proposal. The purpose of the tool is to obtain user data sets that can later be used in the summarization tools. The data sets will come from product reviews, blog posts, and forums, and it is important that the tool be able to support each of these kinds of websites.

The web-crawler should obtain the following attributes from each review: Product Name, Review Title, Review Body, Helpfulness Count, Star Rating, Review Date.
Blog posts and forums would require similar attributes to be collected that would approximately fit into the attributes described above for reviews.

An overview of each attribute: The product name is the name of the product found from searching the web for the desired search string. The review title is the subject line associated with individual reviews. The review body is the main portion of text from the review and is the primary piece of data.

The helpfulness count is the thumbs up found on many reviews. This gives us an integer value of how helpful humans find the specific review so we can pay more attention to these significant reviews. The star rating is how high the reviewer rated the product; we normalize this to an integer, one through five. Finally, the review date is when the review was posted, so that we can see how current each review is.

In addition to taking data sets from multiple sources, the web-crawler must also be capable of storing them in a MySQL database. This may include pre-processing, depending on later design decisions, to normalize the data collected from different sources so they all fit into the database.

As well, it is important that the web-crawler comply with all website usage policies so it is not flagged as a DDOS attack by third-party web servers. Therefore, for any website with DDOS protection, the web-crawler must be able to automatically throttle the crawling so that it won't be turned off. While this may slow down crawling for certain websites it is more important that we are able to continuously, autonomously, crawl.

Database

To store the majority of the data the web-crawler would be gathering, we identified the need for a database to store both review and summary data. For the review data, each database element needs to reference a given review of a product.

For proper organization, each element will need to contain the review body, title, helpfulness, rating, source, product, and other potential attributes. These extra attributes would change based on the source website of the review. For example, some websites have “helpfulness” ratings, while others do not. Therefore all the collected data will have to be normalized so it is compliant with the database.

Along with the raw data from the crawler, the database will also need to be able to store the summarization results. This includes not just text but also any tables or images produced. This will allow for easy access and displaying of the product summary. Finally the database will need to be able to interface with the web application so that the data can be accessed and displayed. This is also important for the user experience, as we expect users will not want to manually interact with the database.

Summarization Tools

Brief summary of requirements:

1. Clean the data removing extra, unimportant, data
2. Summarize the reviews using various techniques
    a. Extractive summarization
    b. LDA topic modeling
    c. Abstractive summarization (stretch goal)
3. Should interface with the database for data retrieval and storage.

Details on these requirements:

To satisfy the project proposal, we needed a summarization tool which could present the large amount of collected data in a more manageable, quick and easy to understand format. The toolset should be modular enough to take in the data from the database run and return the new summarized data in a format which can easily be put back into the database for storage.

The tools should produce clean summaries without any superfluous information. To accomplish this, we need to clean the raw data collected before summarization. Our primary concern is removing extraneous words and inordinately brief reviews that do not actually contribute to the discussion of the desired item. For our purposes, this means any review with less than ten words. We expected other data cleaning techniques would be needed to improve the quality of our summaries without impacting their semantic meaning.

With regards to the types of summaries we need to produce, our client identified two major requirements. First is an extractive summarization tool, which uses the corpus (group of documents) to generate summaries that are in fact a subset of the

original corpus. That is, any phrase or sentence found in the summary existed somewhere in a review, and was simply extracted.

The second type of summary tool we need is a keyword summarization, with would use topic modeling to identify and group keywords to summarize the given corpus. For this purpose, we were asked to use Latent Dirichlet Allocation techniques, which uses generative probabilistic models to perform topic modeling and identify keywords. For more information on this technique, please refer to the “Additional Information on Summarization Techniques” section in the Developer’s Manual.

If time permits, the client has also requested a second summarization tool that utilizes abstractive summarization techniques, which rely on neural networks and machine learning to generate brand-new text summarizing the dataset. This is a much more complicated tool, which will require a great deal of time and effort to implement and configure, so it is considered optional to the success of the project.

Web Application

The web application exists to provide the user with a natural interface with which to run and use the other tools that comprise the project. There are multiple tasks the user needs to be able to perform:

One is submitting a request to see reviews for a certain product, which should activate the web-crawler. Second is to submit a request to generate summaries for a scraped set of review data, using one or more of the summarization tools provided in the product. Third, the user needs to be able to access both review and summarization information for both newly generated requests and past requests. Previously generated summaries should be stored for quick access, but also provide a timestamp so the user can identify the freshness of the summary.

Design / Implementation

Web-crawler

For the base of the web-crawler, we used the open-source tool Scrapy, “an application framework for writing web spiders that crawl websites and extract data from them”, as explained on their documentation site [1]. It is implemented in Python, which allowed for easy extension of the framework to suit our specific web-crawling applications, and allows for multi-platform support. To illustrate Scrapy’s execution, please refer to Figure 1, from the Scrapy documentation at doc.scrapy.org [2].

Scrapy uses object classes called spiders, as shown in Figure 1 item 7, to allow for customized crawling per website. As mentioned in the Requirements section, automatic throttling of the spiders was required for them to be friendly to the sources being crawled. If the spiders were not friendly, then they would quickly be blocked from the source and the crawl would fail. The automated throttling was handled by the scheduler shown in Figure 1 [2].

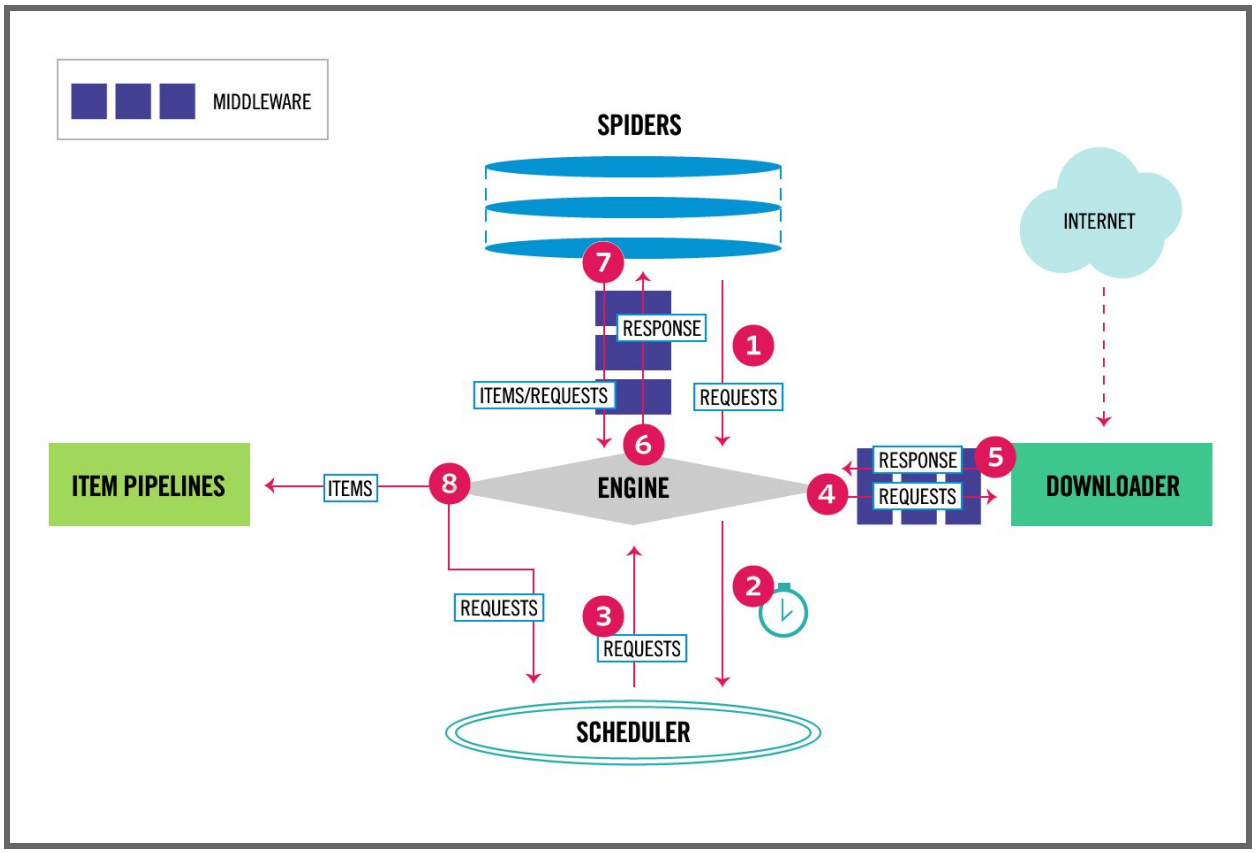

Figure 1: Scrapy Architecture Diagram [2]

While Scrapy offers customizable pipelines to filter and preprocesses data, this feature that was not used due to additional complexity and overhead that extended beyond the scope of our project. Instead, we used external Python scripts to preprocess the data after the crawling was complete. The use of these external preprocessing scripts allowed for each spider to have a customized preprocessing function to apply before being inputted into the database for later use.

The output from the Scrapy web-crawling is saved as a JSON file with the search string as the file name. Throughout the rest of the program, we continue to use this naming convention with additional "-keyword" tags. This allows us to save temporary files before the raw data and summarized data is entered into the database, and was essential for maintaining test-driven development.

In our preprocessing python scripts, the data was processed and reformatted to another JSON file that is properly formatted to fit the summarization toolkit and database. The main script, database-preprocess, takes the raw JSON file generated from the web-crawler and formats it for the database by removing all Unicode special characters from the text strings, extracting helpfulness counts or percentages, and extracting a star rating into a numerical representation. It operates as follows:

To begin, non UTF-8 characters are removed from all of the text, as the database expects all string text to be in UTF-8 encoding. Next, the values collected for review-ratings and review-helpfulness are converted from strings to integers. The integer representation is important not only for storage in the database, but also so these values can easily be used for sorting and comparison.

The review data is then sorted and organized by the (now numerical) helpfulness value so that the reviews humans thought were most helpful would be weighted more valuable for summarization later on. All of this cleaned data is then saved as a new preprocessed JSON file to be entered into the database. The new preprocessed file is saved as [search string]-pp..JSON.

Database

Based on the above requirements for our data storage, the team decided to use the MySQL relational database system. Currently there are separate tables for the sources, products, and reviews. The relational database model allows us to add specific details to a product or source, for example, without manipulating any rows in other columns, leaving the data independent.

For future use, using a relational database also allows for the ability to add new tables with relation to a review, source, product, or any other dataset that may be needed with only a foreign key and not large updates to the entire database. For example, we could add a related products list to each product without interfering with the reviews connected to the product. This allows for easy database manipulation and growth.

Summarization Tools

After the raw JSON file is preprocessed and the new JSON file has been generated, the summarization can begin. Instead of fetching the preprocessed file from the database, we implemented a time-saving measure in which the summarization tools also take the newly generated preprocessed JSON file before it is sent to the database. It is much more efficient to pass the file directly to the summarizing tools, as opposed to uploading and then immediately pulling the file from the database. In addition, the summarization tools require the same preprocessing as the database to operate, and the organization done by the preprocessor actually improves performance.

The complete summarization toolkit is implemented in the script GensimUse.py. The design decision to have multiple summarization techniques enfolded into one script file was a pragmatic one. Namely, we would get much better performance if we limited the number of script calls required by the web application. It also helped efficiency by reducing duplicate function calls on the data, since each tool has to initialize the dataset in the same way.

GensimUse.py takes in a single string parameter which is the search string that is the foundation of our naming convention. It uses this parameter to load the preprocessed JSON file. More information on using the script can be found in the Developer's Manual later in this report.

Before any specific summarization techniques are applied, there is an initialization phase the script must complete to prepare the data. It creates our stopword list at this time. "Stopwords" are words in the review text that we will ignore during certain summarization tasks. A good example is common words such as 'the, a, as, for, am', which appear frequently in written English but carry very little semantic meaning in a summarization context.

Our program's stopword list is primarily taken from a Python open source library called Stopwords [3]. We also include a few of our own stopwords to improve on this list, as we identified the need for more stopwords with our testing. For example, testing showed that we needed to include the words included in the search string. This was an important step because when we are performing summarizations, it is intuitive that the product name is an important keyword. If not filtered out, it will appear frequently in nearly all the summarizations, which is unhelpful and redundant.

To continue the initialization process, the script then begins building six different corpora for latter summarization. Each corpus is created using only reviews which had a review-helpfulness greater than zero. This means that at least one human found the rating helpful on the source website. That helps to prevent the usage of unhelpful reviews whether they are positive or negative.

However, testing showed an issue if the corpus is created and there are not reviews with a helpfulness greater than zero. In this case, we will then build the corpus with all the present reviews to get some data for the product. This is undesirable, but preferable to no summary at all, especially since we do have some data to work with.

The six different corpora are for the five different rating levels, one through five, along with a full product corpus containing reviews of all rating levels. Our rationale

behind this division of individual rating levels is so that users can not only get summaries of general opinions, but also a sense of what people who rated the product negatively or positively had to say, and how that contrasts. We believe this approach gives the user the widest range of useful information that can be accessed and understood in a very short time.

After each corpus is constructed, more error checking is required to ensure that there is enough information present for summarization. It is important that the summarization tools not run on an empty dataset! For example, if a product did not have any reviews with a two-star rating it would not generate a two-star and none of the summarization output files for this star rating would be produced.

After the corpus is built and validated we can begin using our summarization tools. We made use of Gensim, “a free Python library designed to automatically extract semantic topics from documents, as efficiently (computer-wise) and painlessly (human-wise) as possible.” [4]

Gensim provides several features for summarization of various types. We are making use of two extractive summarization tools and one LDA modelling tool. The first extractive tool is the general “summarize” command, which extracts words and sentences from a corpus to generate an easy-to-read summary. The second tool is a keyword feature. This tool works similarly to the “summarize” command to extract important keywords from a corpus. Gensim also provides an easy to use LDA model which we use for topic generation.

Gensim’s extractive summary body generator takes in the corpus and can create a summary of the inputted documents using TextRank algorithms. (For more information on these algorithms, please see the section “Additional Information on Summarization Techniques” in the Developer’s Manual.) The function can limit the summary length by a percentage representing the ratio of the corpus word count to the summary word count. For example, if the corpus had 1,000,000 words and the ratio was 0.2 then the resulting summary would have 200,000 words. In addition, it can also take a hard word count capacity value which takes priority over the ratio. This ensures the summary length remains reasonable for very long corpora.

The keyword extraction also uses the TextRank algorithms. However, to prevent unwanted words appearing in our list of key words we must utilize our previously generated stopword list to remove unhelpful and irrelevant words from the corpus. Additionally we perform a process called “lemmatization” on the corpus. This merges duplicate and similar words. For an example, view the diagram in Figure 2, from an article by computer scientist Scott N. Dobbins [5]. This ensures that we generate a more diverse selection of keywords without sacrificing any meaningful semantic information.

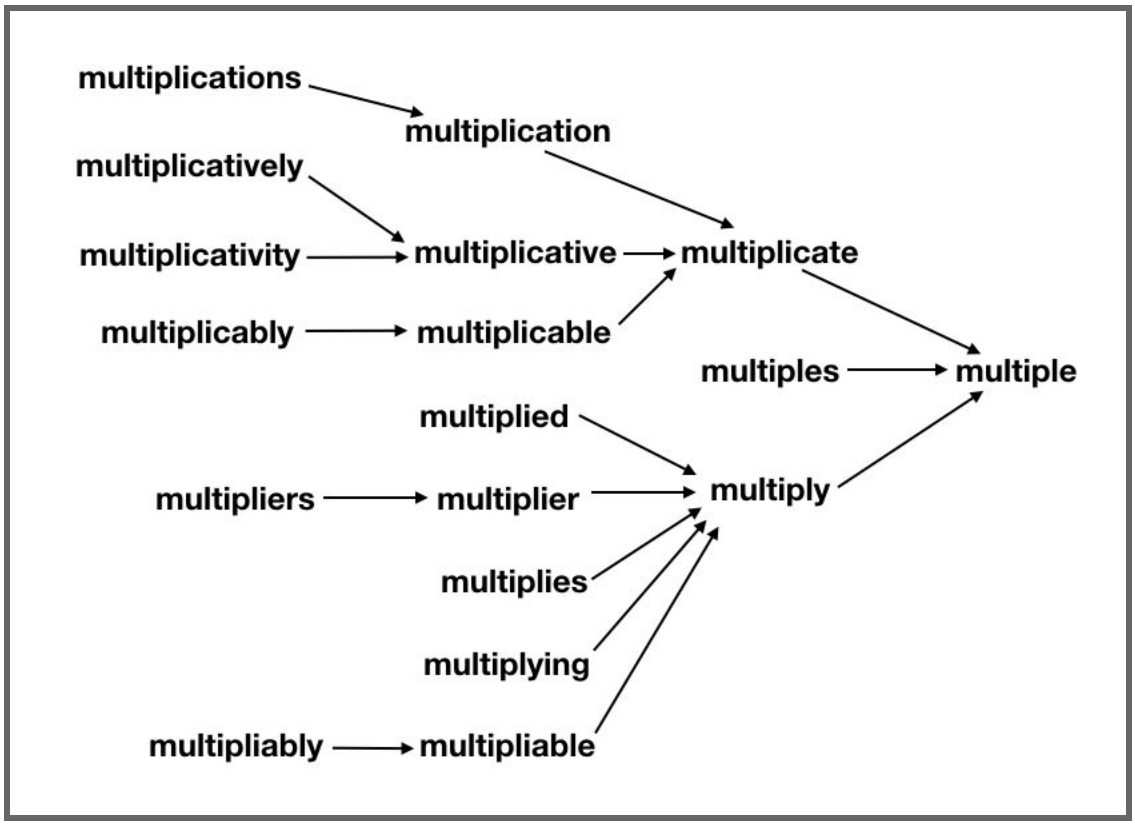

Figure 2: Lemmatization Example [5]

The final summarization tool is the LDA topic modeling [6]. Similar to the keyword extraction, we first remove stopwords and lemmatize the corpus. We then input the cleaned corpus in Gensim's LDA model and let it generate topics using the Latent Dirichlet Allocation algorithm. For a description of this algorithm, please see the section "Additional Information on Summarization Techniques" in the Developer's Manual.

The script then outputs the summary body and keywords into a .JSON file named [search string]_[summary rating]_summarized..JSON, in accordance with our established file naming conventions. The [summary rating] field is the numerical rating of the corpus used to generate that specific summary. Note: While the summary rating for the entire product summary is 0.0, this is purely for naming purposes and does not reflect on the quality of the product.

The tool then generates word clouds from both the generated keywords and the topics produced by the LDA algorithm. To create these word clouds, we use an open source Python library called WordCloud. The purpose of these figures is to provide a visually appealing way to view two of our three summarization techniques in a quick and easy fashion. This provides maximum user understanding with the least required mental strain, making our summarizations more affordable to users.

The keyword word cloud picture is named [search string]_[summary rating]_keywords.png. Each topic word cloud picture is named [search string]_[summary rating]_topic_[topic number].png. Each corpus generates several

topics and the topic number indexes each of them. (Please note that the number order is arbitrary and does not reflect the value of the topic!)

Assuming the product had enough reviews for each ranking, the script will output the following: Six summary JSON files, six keyword image files, and six groupings of individual topic images where each grouping contains the desired number of topics. All of this summarized information is then stored in the database under the product that was originally crawled along with the preprocessed review data. All of the generated information can then be browsed at the user's leisure by using the web application.

Web Application

The Web application is implemented with Entity Framework .NET Core 2.0 [7]. The .NET Core 2.0 framework is the newest of Microsoft's .NET frameworks and is promised continued support and new functionality for a long time to come. For all database interactions, we are using Entity Framework because of the ease with which it interacts with the .NET framework. Through the use of lazy loading and LINQ, we are able to access and manipulate the database very quickly [8]. This is especially advantageous for our uses as we need to be able to add many thousands of elements into the database at once.

In addition, Model-View-Controller (MVC) structuring is used. Each table in the database is represented as a model. Additional models are used for passing data to and from the different views. This allows for incredibly unconvoluted and secure data transfer. The controllers are used to handle all backend logic and data manipulation. They pass the models to the views, manipulate the data in the models, and handle whatever the views then pass back. Redirection is handled in addition to adding values to the database. In addition, all logic related to error handling should be in the controller [9].

This results in a structure where the controller can manipulate the view and the model, and the view can manipulate the model, and the controller can access that manipulated model, as demonstrated in Figure 3 [10].

The Web Application is deeply integrated with both the web-crawler and the summarization toolkit. All scripts required for crawling and summarization are embedded in the application, allowing for easy access. This organization also provides a seamless pipeline between all aspects of this tool, giving the user a seamless, simple experience for crawling the web and generating summarizations of products.

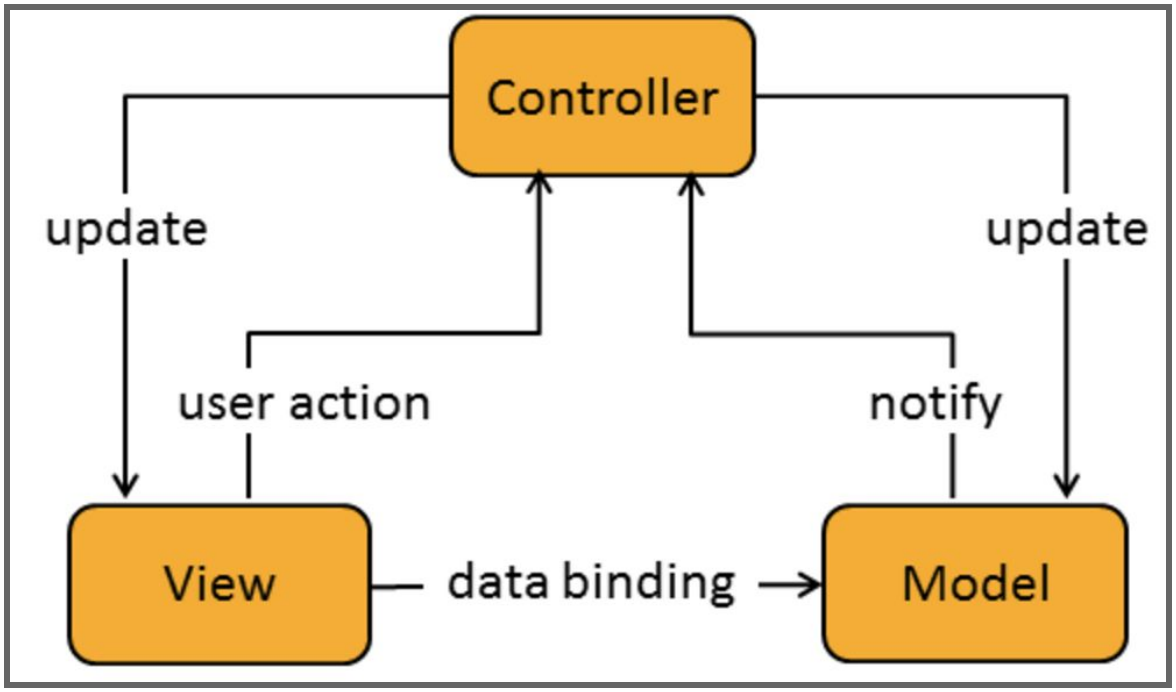

Figure 3: MVC structure [10]

Testing / Evaluation / Assessment

Web-crawler

Our initial goal was to scrape the data for reviews from a specific URL. We evaluated our results on two main qualities: completeness and speed. Download speed is important to monitor, as it can be severely limited due to website throttling. The web-crawler must be friendly to the website so that it does not affect regular users. If the web-crawler was not friendly, then the website host would deny availability temporarily. After getting one webpage to work, we added more functionality and tested these conditions incrementally.

Completeness was easy to verify, by comparing the number of reviews given on the webpage to the number of reviews successfully scraped by our Scrapy-based web-crawler. Unfortunately, we have not found an efficient way to validate the content of each review, as there is too much body text to validate manually. However, everything seems to be in order, and the amount of text we are scraping seems in line with what is present on the review website.

However, when the number of reviews is below fifty for a particular product, the web-crawler is unable to detect and collect properly. Therefore, the minimum required reviews for a proper crawl needs to be greater than 150. This issue was not a large concern as the summarization tool would require more than fifty reviews in most cases.

Ultimately, we were able to create a web-crawler for Amazon and Walmart. The Amazon web-crawler works with any product but particular care does need to be taken when choosing the input string since more than one-hundred and fifty reviews should be present for the first product in the results page; note that “Sponsored” products are ignored. Unfortunately, the Walmart spider was not as successful due to Walmart blocking Scrapy from doing any type of crawling, creating critical issues with dynamically obtaining review information.

Database

Since the database performs a relatively simple set of tasks, the main testing is ensuring that the database has been set up properly. For this process, we first had to ensure that the database was correctly designed and constructed. Design checks were accomplished through communication with other team members to ensure compatibility with the other tools. To check constructions, we manually evaluated the database, making sure all the desired fields were set up properly and connected through the correct keys.

The next step in testing the database was to try each of the important operations, such as adding and removing elements from the database, to ensure correctness with our data, and that no information was being lost. We used dummy data for this and performed comparisons on the data before and after it was stored in our database.

After we had the storage operations in place we began testing fetching, specifically on individual reviews and products. After we had all of the essential

functionality in place, we tested some of the more advanced functions: First was making sure it connected to the web-crawler properly. Second was testing batch addition by adding all the reviews for a product all at once. We needed to ensure this process was quick and efficient so as to not slow down the flow of the whole toolset. Over all, most functionality testing is done by the web application as the database is not an efficient place for error checking.

Summarization Tools

Testing the summarization tools was an incremental process that took place concurrently with development. First, we took sample data collected from our web-crawler, using the rare technologies gensim summarization tutorial to test understanding of the Gensim library [4]. After that, we progressed to using our own test data to summarize the reviews we collected and to evaluate them based on the following criteria: Coherence, Coverage, Overlap, Order.

Coherence is the overall sense of the summary from one sentence to the next. The coverage examines how much of the information in the original body of text is still represented, and how much meaning is lost. Overlap is how much information is repeated in the summary. Order is how much sense the arrangement of sentences makes.

Evaluating our summarization was very subjective and not very quantifiable. However, using the previously mentioned criteria we could test and improve our summarization. For the extractive summarizer, the Gensim tool we were using allowed us to get excellent coherence and order from the beginning.

However, since we started off with just a ratio to determine the length of the resulting summary we got a surplus of overlap in the summary as it repeated phrases that appeared across multiple reviews. The coverage was also too specific, causing the summaries to be too long, including unnecessary details. To solve this problem we included the word cap which kept the coverage and overlap in check. You can see the effect of the word cap on long and short product bodies in Table 1, generated from our results.

Product Name	Rating value 5.0 Corpus Word Count	Summary Word Count with Ratio	Summary Word Count with Ratio and Word Cap
Nintendo Switch	83638	16727	500
Switch Case	5687	1137	500
Dell Inspiron 13 500 2-in-1	2010	402	402

Table 1: Corpus and Summary Length Comparison

Since the keywords and topics both produced lists of words and required the same input, it was possible to evaluate them together, as most issues could be addressed by changing the input to the tools. Figure 4 is an earlier word cloud produced from keywords collected from the Dell Inspiron reviews. As you can see, it contains many words that do not add to or explain the product. The only indicator that this is for any specific product is the word laptop in the top left corner.

This demonstrated the dramatic need for preprocessing these inputs and cleaning the corpus used for the keywords and topic summarization. After implementing stopwords and lemmatization, as described in “Design / Implementation” above, our topics and keyword extraction were significantly improved. This allowed us to produce the second word cloud, shown in Figure 5. It is significantly better than the previous one (Figure 4) in terms of coherence and overlap.

You may notice that for this example, there are still a few words that should potentially be added to our stop list, such as “also”, “look”, and “take”. However, we do not want to include too many words in the stopword list, because it is important that this tool be usable on a wide range of products. Continuing the example, when searching for reviews on a pair of binoculars, the word “look” may be extremely relevant to a topic. While our stop list is fluid and constantly being improved upon, it is essential that we be careful to not overfill it.

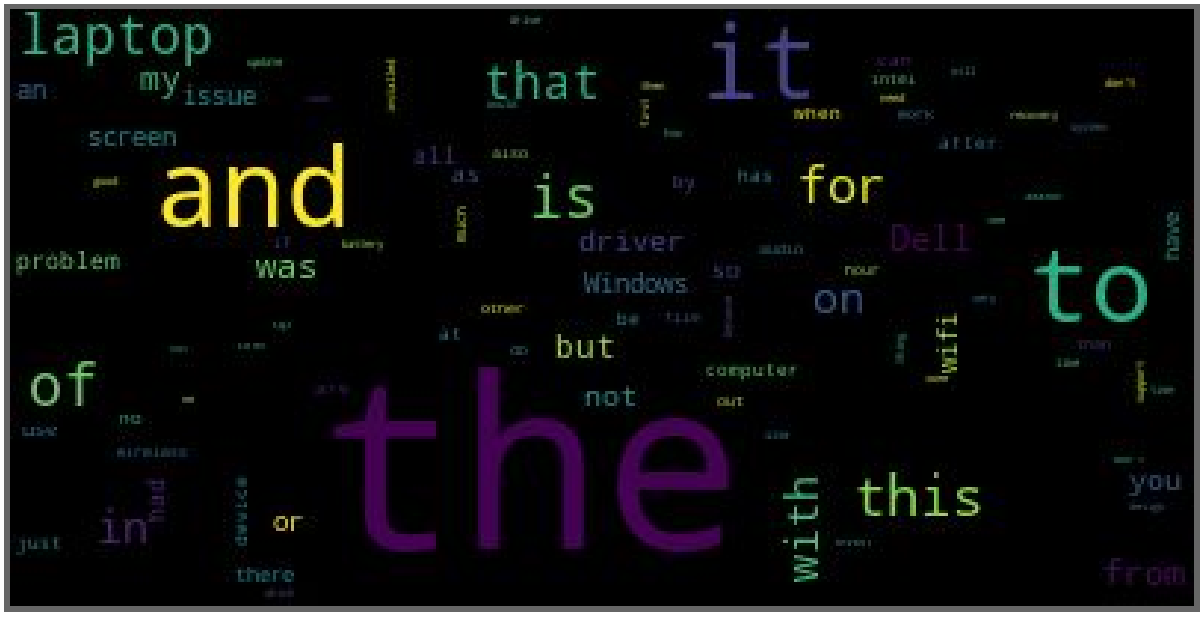

Figure 4: Keywords created before removing stop words

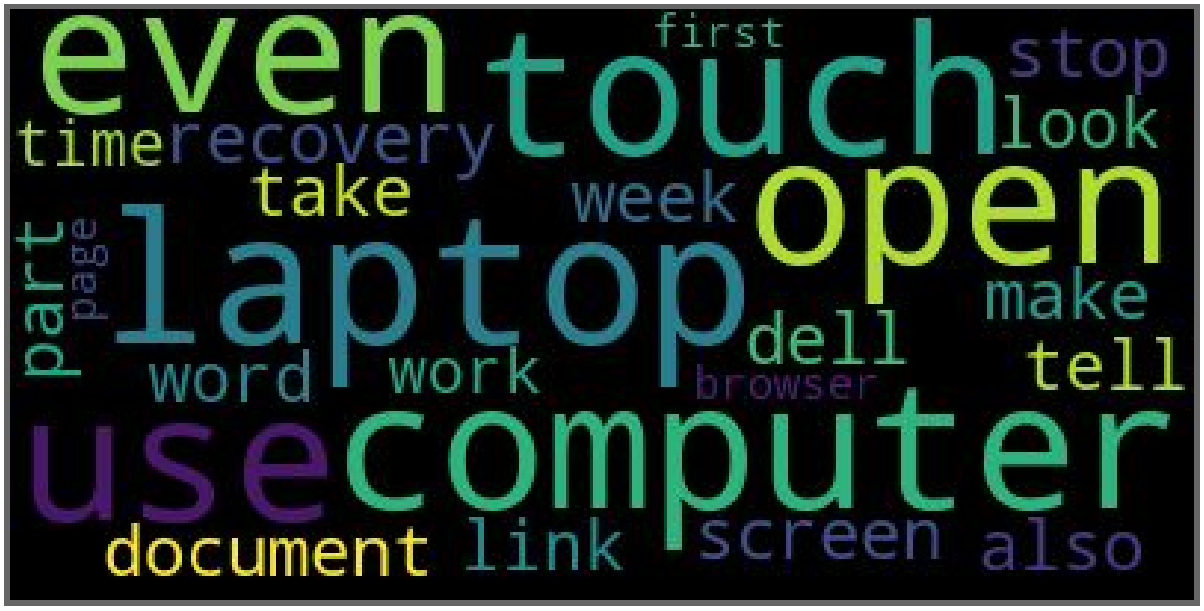

Figure 5: Keywords created after removing stop words

Web Application

To test the web application, we have performed a great deal of trial use and simulated user testing. We are practicing incremental development. First a feature is added, then we ensure that the feature works as expected through the simulated user testing mentioned above. In addition, we are also using Unit Tests for each of the features. The .NET framework allows for generation of many of the tests requiring test data that we make up [11]. This allows us to know the application has remained intact through every new build, giving us more freedom to make changes without the concern of compromising the application's functionality.

This verification of integrity is important, since the web application is the user's sole point of access to all of the tools in our project. Therefore, it was important that each individual tool be attached properly to the web application, so the user has a way to use them!

Furthermore, whenever any feature is added, edge cases must be thoroughly checked. For example, when implementing the search feature, both null and extremely long entries are thoroughly tested along with possible SQL injections and XSS attacks. Error handling for invalid parameters being passed into views is also tested, this allowed us to test a large range of different types and edge cases in those types, ie SQL and XSS attacks.

Each connection between the web application and a tool required its own testing. When first connecting the web application to the database, we tested the connection thoroughly by using a variety of different inputs. We paid special attention to finding unhandled exceptions for overly large or small inputs, then implementing ways to handle them.

The main testing of the database involved checking entity frameworks binding to the models. This required a significant amount of time, and tests checking that each database attribute was mapped one to one to the respective model attribute.

When connecting the web-crawler, thorough testing was required for running the different scripts required, as well as parsing the JSON files generated for input into the database. It was important that all requirements for the web-crawler be met, and that the user be able to easily initiate a crawl without errors or freezing the application.

Finally we connected the summarizing tools, including automatic retrieval of the data to be used. We did final checks on the functionality of the web application through more simulated user testing. Every operation was performed multiple times, making sure all of the components were interacting properly. To ensure that no information was lost during database storage, we performed diff checks on the expected output and actual resulting output.

Future testing would involve more focused user testing, focusing on HCI and user experience principles to evaluate the web application for usability, learnability, and affordances. The goal would be to make sure the layout and design of the interface is intuitive and easy to use, so that those who are not computer scientists or web developers could search and examine product summaries with ease.

Users' Manual

Setting up the Web Application

1. Install Python 2.7
2. Go to http://aka.ms/vcpython27 and download the C++ compiler for python 2.7
3. Run the following commands in powershell to install required Python libraries:
    a. "pip install gensim, wordcloud, pattern, stop-words, unidecode"
    b. "pip install scrapy, scrapy-fake-useragent, pypiwin32"
4. Install Visual Studio Code 2017 15.6.7, Community free version is acceptable,
    - See Figure 6.
5. NuGet Package manager handles all the dependencies for the solution. These packages will be installed automatically upon opening the solution. These packages consist of:
    - Microsoft.AspnetCore.All v2.0.3
    - Microsoft.NETCore.App v2.0.7
    - Microsoft.VisualStudio.Web.Code.Generation.Design v2.0.3
    - MySql.Data v8.0.11
    - MySql.Data.EntityFrameworkCore v8.0.11
    - Newtonsoft.Json v11.0.2
6. Press f5 to run solution

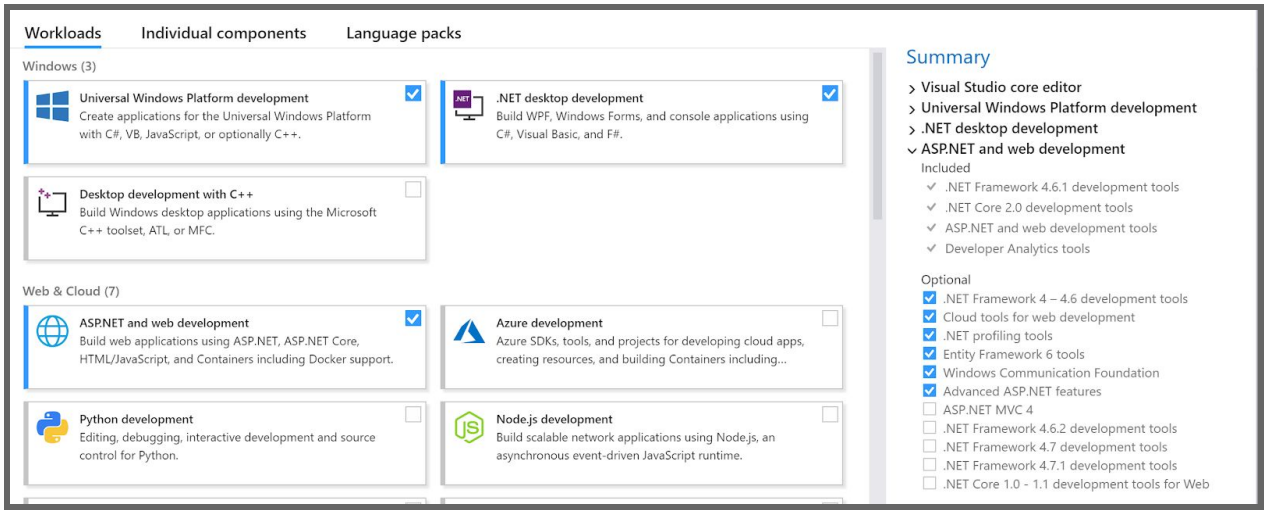

Figure 7: Visual Studio Packages

Using the Web Application

When you first navigate to the application, the home location is the products page, which contains a list of already searched products. On this page you can browse recently crawled product reviews, crawl and summarize a new product, or view the summarization of an already crawled product. Each previously crawled product contains a name, source, and a button to view the summary. At the top of the page there is a search bar which will allows the user to search a new item.

When the search text field has been filled with the desired search text, the search button must be selected. The search button begins the crawling and summarization. Depending on the size of the products review set, the crawling can be as short as 30 seconds, and as long as several minutes. Once the crawling and summarization completes, the page will be reloaded containing the new product that was found and summarized.

Selecting the summarization button highlighted in Figure 7 on a given product will display the summarization and word clouds containing the keywords for a product. Summaries will be displayed with the summaries stored in descending order with the overall summary at the bottom as displayed in Figure 8.

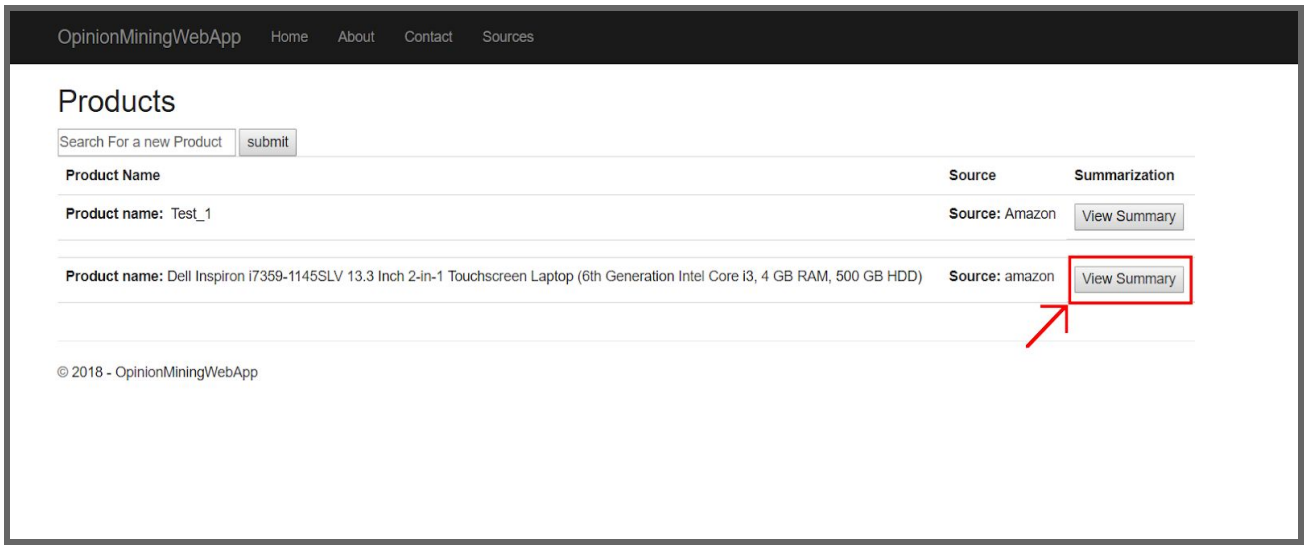

Figure 8: Individual Product selection screen

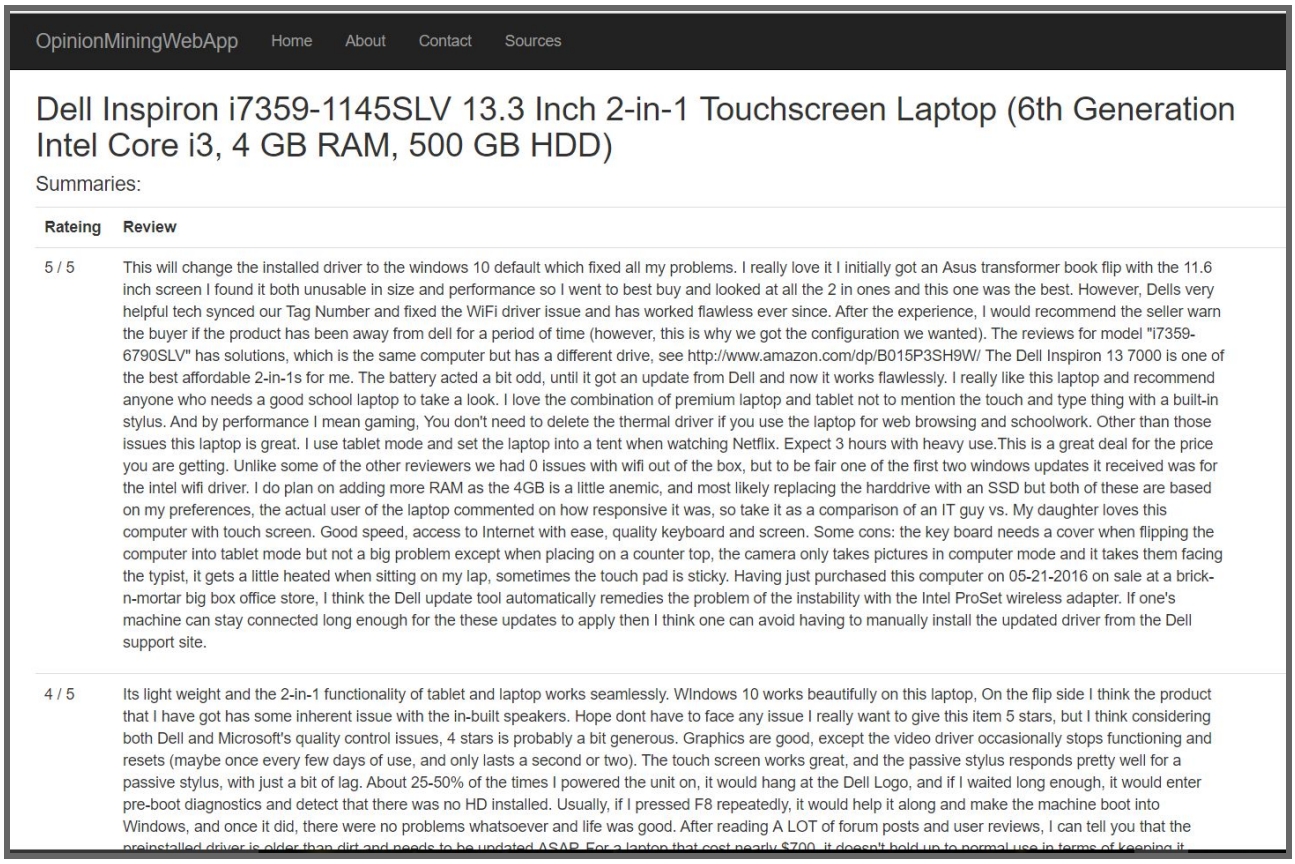

Figure 9: Selected Product Summarized results

Browsing Existing Summarizations

Beneath the search bar, the homepage displays a list of currently existing products that have been summarized by our tool. Scroll through them and click on the summarization you would like to view in detail. Alternatively, enter the name of a product. If a summarization exists, it will be highlighted.

Generating New Summarizations

To generate a new summarization, begin by entering the name of a product into the search bar. If a summarization already exists, it will be highlighted. If not, there will be an option to “Generate New Summarization”. Click this button and wait for a time estimate to be displayed. You can now freely navigate away from the web application, and the results of your summarization will soon be uploaded to the website. They can then be accessed in the same manner as “Browsing Existing Summarizations” above.

Developer's Manual

Web-crawler

The web-crawler is built using the open-source Python framework Scrapy. Scrapy uses Python object classes called spiders to crawl through the web. Spiders determine how a certain website, or potentially multiple websites, are going to be scraped [12].

The spider execution cycle goes as follows:

1. Go to specified URL and download HTML page.
2. Callback function is called and user defined behavior can begin.
3. Data from the HTML page is selected using Selector tags, similar to jQuery [13].

To create a brand new web-crawler you will just need to follow the basic structure shown in Figure 9 [12] and described below.

```
1   import scrapy
2
3   class MySpider(scrapy.Spider):
4       name = 'example.com'
5       allowed_domains = ['example.com']
6       start_urls = [
7           'http://www.example.com/1.html',
8           'http://www.example.com/2.html',
9           'http://www.example.com/3.html',
10      ]
11
12      def parse(self, response):
13          for h3 in response.xpath('//h3').extract():
14              yield {"title": h3}
15
16          for url in response.xpath('//a/@href').extract():
17              yield scrapy.Request(url, callback=self.parse)
```

Figure 10: Scrapy Example Code [12]

Structure Description:

- ***name***
  How Scrapy will be able to determine what the spider is called. The name should be unique and give enough information to know the type of spider it is.

- ***allowed_domains***
  An optional field that restricts where the web-crawler is allowed to proceed during a crawling procedure.
- ***start_urls***
  The sources that the web-crawler beings to crawl for the desired data. *start_urls* is able to contain multiple sources, but in general most sources that we are using have very different layouts and therefore require a different web-crawler to retrieve data.
- ***parse***
  Default function called when a webpage is retrieved after a request is sent to the *start_urls* defined. In the parse function, user defined behavior is defined to dynamically search for particular tags or IDs in the HTML response. The selector tags above are using Xpath format, but CSS format can also be used. The CSS format is similar to jQuery selector for DOM objects. Looking at the Amazon and Walmart cases for other guidelines will allow for additional sources to be added quickly.

Additional attributes of a spider can be defined and are gone into more detail in the official spider documentation on the Scrapy website [12].

Database

The database is a MySQL relational database. The database currently contains 5 tables. A product contains a list of reviews and a list of summaries. A summary contains a list of keywords and each review has a source.

- **Server IP:** 128.173.49.55
- **Port:** 3306
- **Database:** user_reviews
- **Uid:** cs4624_s18
- **Password:** DLRL_2030

server=128.173.49.55;port=3306;database;uid=cs4624_s18;password=DLRL_2030

- **keywords:**
  - pkKeyword, INT, UNSIGNED, PRIMARY, AUTO_INCREMENT: Primary Key
  - SummaryID, INT, UNSIGNED, FOREIGN KEY: Foreign key from summaries table. There is a one to many relationship where a summary can have many keywords.

    - KeywordText, TEXT: Text field for a keyword found in a summarization.

- **products:**
    - pkProduct, INT, UNSIGNED, PRIMARY, AUTO_INCREMENT: Primary Key
    - Name, VARCHAR, LENGTH = 500: Title of a product
    - SearchString, TEXT, ALLOWNULL: String entered when the value was initially searched.

- **reviews:**
    - pkReview, INT, UNSIGNED, PRIMARY, AUTO_INCREMENT: Primary Key
    - SourceID, INT, UNSIGNED, FOREIGN KEY: Primary key of a source element. There is a one to many relationship between source and review. One source has many reviews.
    - ProductID, INT, UNSIGNED, FOREIGN KEY: Primary key of a product element. There is a one to many relationship between product and review. One product has many reviews.
    - Rating, INT: Rating of a product by the reviewer
    - Body, TEXT: Content of the review
    - Title, TEXT: Title of the review
    - Date, DATETIME: Date the review was posted
    - Helpfulness, INT: Helpfulness rating of the review

- **sources:**
    - pkReview, INT, UNSIGNED, PRIMARY, AUTO_INCREMENT: PrimaryKey
    - Url, VARCHAR, ALLOWNULL: URL of the source.
    - Name, VARCHAR: Name of the source.

- **summaries:**
    - pkReview, INT, UNSIGNED, PRIMARY, AUTO_INCREMENT: Primary Key
    - SummaryText, TEXT: Summarized text.
    - ProductID, INT, UNSIGNED, FOREIGN KEY: Primary key of a product element. There is a one to many relationship between product and review. A product has many summaries,
    - Rating, INT ALLOWNULL, DEFAULT NULL: Summary rating collection.

Summarization Tools

The summarization tools are all contained in a single Python file named GensimUse.py. The code has four functions total: three helper functions for reading data and summarizing along with a main function; these are: main, getDocuments, getTopics, and keywordSummaryExtraction.

main:
Parameters:
searchString <String>: the inputted string that was used for searching
Output: saves several files for the summary
Function: Begins by loading the [search string]-pp..JSON file and stop list. Then begins creating corpora for the whole product and the five rating levels using the getDocuments function. Then, using the getTopics function, it creates the topics and generates word clouds for each topic. Then it uses the keywordSummaryExtraction function to generate the summary and keyword list which is then saved as a JSON file along with a PNG for the keyword word cloud.

getDocuments:
Parameters:
rating <Integer>: desired rating to build the corpus from
data <Loaded JSON dictionary>: dictionary loaded from the [search string]-pp..JSON file
Output: **documents** <Array of Strings>: built corpus from review data
Function: Builds a corpus from the loaded preprocessed data using only the reviews with the desired rating. If the rating is 0.0 it will load all reviews regardless of rating. Begins by attempting to find reviews with a helpfulness greater than zero; however, if no reviews can be found for the inputted rating it will load any review regardless of helpfulness.
NOTE: There is a DOCUMENT_CAP variable at the top which can be useful when running the script on weaker hardware. It is set to 1000 by default but can be increased on better hardware.

getTopics:
Parameters:
documents <Array of Strings>: acquired from the getDocuments function
numtopics <Integer>: desired number of topics
numwords <Integer>: desired number of words per topic
stoplist <Array of Strings>: stopword list
Output: **topics** <Array of Tuples>: each topic tuple contains the “raw-topic” from the LDA model and the “cloud-text” which is used for word cloud generation.
Function: Uses LDA topic modeling to create the topics array which can be used to produce the word clouds.

keywordSummaryExtraction:
Parameters:
documents <Array of Strings>: acquired from the getDocuments function
ratio <Float>: ratio of words in documents to summary word count
word_cap <Integer>: cap on summary words
keyword_count <Integer>: desired number of keywords
stoplist <Array of Strings>: stopword list
Output: **summary** <Tuple>: the first element is the "summary" which contains a string that is the summary body, the second element is the "keywords" which is an array of strings which are the keywords.
Function: Uses extractive summarization to produce a summary body and a list of keywords.

Web Application

Using Entity Framework Core 2.0 to connect the database with the web application, each database table, has a model represented in the web application. Each model has a one to one mapping of each database field in the respective database table. All querying, adding, and other manipulations of the database are entirely done through LINQ.

The Web Application is built using the MVC structure. Inside of the web application path is **OpinionMiningWebApp/OpinionMiningWebApp**. Not including script files relevant to styling, all code relevant to the web app and project exists here.

/Controllers/

1. **ProductController.cs:** This is the controller for the home page of the web application that displays all the products.

- **CrawlPending** is an IActionResult method that is called when searching a new product is selected on the homepage; this method calls a method RunScrapy. Upon completion of RunScrapy, a webpage will load, informing the user that their product has been scraped and summarized.
- **RunScrappy** takes in a single string argument which is the search parameter the user used. This string is parsed by removing all white space. This new processed string is now used for the base of all files generated in relation to this product. For example: "Dell Inspiron 2-in-1" -> "DellInspiron2-in-1". The search string is also passed into the database for future retrieval. Crawling is begun using a RunCmd method which simply takes in a working directory, and a string array of commands to be run in sequence. This is done first for the crawling, then for the preprocessing of the JSON, and then for the summarization script. Finally, after this completes, adding the files to the DB incurs through AddJsonDataToDB.
- **AddJsonDataToDB** takes in a single string which is the processed search string, and loads up all the files that were generated. Using JsonConvert.DeserializeObject, the JSON files are mapped one to one to a model replicating the JSON object in the file. The product name and source are both

inserted into the database to receive the primary keys. These are then used to insert each review with the respective source and product primary key. After inserting the reviews, summaries and keywords follow a similar pattern. The summaries are first inserted and the database saved so that when the keywords are inserted, they can use their respective properly generated summary foreign key constraints.

2. **SummarizationController.cs:** Index is called when “View Summary” is selected on the products page. Index takes in a single argument for the product ID. If this ID is null or if the ID is not a current product that exists in the database, than the user is redirected back to the products listing page.

/Models/

1. **UserReviewsContext.cs:** Represents the database. UserReviewContext contains 5 different elements each representing the respective table and all of which of type DbSet<Model> where model is the respective type for an element in the database.
2. **Keyword.cs:** Represents a single element in the keywords table.
3. **Product.cs:** Represents a single element in the products table.
4. **Review.cs:** Represents a single element in the reviews table.
5. **Source.cs:** Represents a single element in the sources table.
6. **Summary.cs:** Represents a single element in the summaries table.
7. **ReviewInputModel.cs:** Represents a single JSON object from the reviews JSON file that is generated from web crawling. This is used for converting the JSON file into review objects that are added to the database.
8. **SummaryInputModel.cs:** Represents a single JSON object from the summaries JSON file that is generated from summarizing the reviews. This is used for converting the JSON file into summary objects that are added to the database.
9. **SummaryModel.cs:** Represents all elements that are passed to the view for a summary, containing a product, the summaries in relation to that product, and the keywords for each of the summaries.

/Views/

1. **/Products/**

- **Index.cshtml:** HTML file that takes in a ICollection<Product> as the model. All products passed in the collection will be displayed, with a few attributes and a view summary button to display specifics of the summary for that product. In addition there is a search text box which will get an entirely new product crawled for.
- **CrawlPending.cshtml:** This page is displayed when a new product has completed crawling and summarization.

2. **/Summarization/**

- **Index.cshtml:** HTML file for displaying each summary related to a specific review.

/appsetting.json: Contains database connection information as well as the error logging information. The default string connection is:

“server=128.173.49.55;port=3306;database=user_reviews;uid=cs4624_s18;password=DLRL_2030”

Inventory

Date: 5/8/2018
Publisher: Virginia Tech

- Web Crawler
    - Amazon Spider: amazon_spider.py
    - Walmart Spider: walmart_spider.py
- Preprocessing Python Script: database-preprocess.py
- MySQL database, hosted on 128.173.49.55
- Summarization Toolset Script: GensimUse.py
- Web Application, consisting of a Visual Studio Solution and to be hosted on hadoop.dlib.vt.edu
- README.txt, which explains the use of all above files.

Additional Information on Summarization Techniques

Extractive Summarization

To begin with, we use extractive summarization techniques. Extractive summarization is based on the TextRank algorithms, which in turn are based on the PageRank algorithms for searching the web [14]. PageRank is a graph based algorithm which rates webpages based on their connection to other webpages and the value of the pages connect to it. TextRank builds off of this algorithm to rate text in an individual document or a group of documents, also called a corpus, to find the most important words and sentences from the inputted data. TextRank does this by turning the corpus into a graph and running algorithms similar to PageRank and then using the highest ranked parts of the text to summarize the text in various ways. [14]

Latent Dirichlet Allocation

Another summarization technique we use is Latent Dirichlet Allocation (LDA) topic modeling. LDA is "a generative probabilistic model of a corpus." This means the that each document is viewed as a mixture of topics which can be broken down further into words related to the topic. The following algorithm is the foundation of LDA topic modeling taken from Blei, Ng, and Jordan's paper on the subject. [15]

LDA assumes the following generative process for each document w in a corpus D:

1. Choose N ~ Poisson(ξ).
2. Choose θ ~ Dir(α).
3. For each of the N words w_n:
    a. Choose a topic z_n ~ Multinomial(θ).
    b. Choose a word w_n from $p(w_n \mid z_n, \beta)$, a multinomial probability conditioned on the topic z_n.

This generates a groups of 'topics' which contain words related the topic. Each word has a probability associated with it which represents its relation to the individual topic. Each corpus can produce a near limitless number of topics, however too many will begin to produce near duplicate topics, so it is more effective to create fewer topics which more word depth. This is to say, search for less topics while searching for more related words for each topic.

Progress / Lessons Learned

Timeline

Task (Details)	Due
Begin bi-weekly meetings with client on Fridays at 3:30pm.	Entire term.
Web-crawler research (Open source libraries and options)	02/06/2018
Machine Learning research (Open source libraries and options)	02/06/2018
Presentation 1	02/06/2018
Database and web-crawler design	02/09/2018
Database creation	02/16/2018
Basic web-crawler (Connect to a single product website, output data in compliance with database design)	02/16/2018
Data analyzing with mock-data	02/23/2018
Summary Display: Design	02/23/2018
Expand Web-crawler Functionality (Connect to database and multiple product websites)	03/09/2018
Basic Summary Generation	03/13/2018
Presentation 2	03/13/2018
Integrate real-data with data analyzing	03/20/2018
Investigate multiple-document and opinion summarization algorithms	04/01/2018
Presentation 3	04/03/2018
Basic Summary Display (Display pre-compiled summary data)	04/06/2018
Presentation 4	04/24/2018
Additional Features Summary Display (Query back-end to compile new data.)	04/24/2018
Project peer review	05/01/2018

Future Work

Currently we have the foundation for all three of our primary deliverables along with a web application for easy interfacing with our system. Our future goals are to improve upon the functionality of the tools along with the interconnectivity through the web app. For the web-crawler we would like to improve its modularity so that it can work with more review based websites. We would also like to diversify the crawler so that it works with other types, such as blog or forum posts. The database is fully functional and is easily edited for any additional information we wish to store.

The majority of our future work will be invested into improving and expanding our summarization techniques and the web application. We want our summarization to not only use extractive, but also abstractive summarization. This can be done with the library we are currently using, but will require research into these more advanced features focusing on LDI and LSI techniques. There are also mathematical summarization techniques out there which could be easily implemented to give more information about the desired product. For example giving an average word count of reviews could show how many people had a lot to say about the product. Finally we wish to improve the web application for ease of use along with a display feature for the summarization of all products we have collected data on. Any new summarization tools would have to be connected to the database, adding new fields to store new information. This would then have to be connected to the web application to show the new information to the users.

Currently selecting which source to use is a static variable in the web application. Although this is an easy fix, the time for the project did not allow for this feature. This can be done by adding a checkbox for one, or many, specific sources to be searched on the products page. These can be passed in to the running of the crawling and these can concurrently be run, as multiple sources can be crawled simultaneously, while throttling each source the minimum amount for maximum speed.

More error handling is needed for edge cases of the reviews. If too few reviews are available for a product, than summarization is not possible and the user should be made aware of this. Similarly if there are too many reviews this needs to be handled with external file extraction and separation. Depending on the file size, multiple files should be generated and then the files can sequentially be pushed into the database. This would greatly expand the usability of the tool as the products now being able to be searched would expand extensively.

Acknowledgements

Our Client: Xuan Zhang, PhD Candidate, NLP Researcher, xuancs@vt.edu
Our Professor: Dr. Edward Fox, Course Instructor, fox@vt.edu

References

[1] Scrapy Developers, “Scrapy at a glance,” *Scrapy at a glance - Scrapy 1.5.0 documentation*. [Online]. Available: https://doc.scrapy.org/en/latest/intro/overview.html. [Accessed: 02-May-2018].

[2] Scrapy Developers. “Architecture overview,” *Architecture overview - Scrapy 1.5.0 documentation*. [Online]. Available: https://doc.scrapy.org/en/latest/topics/architecture.html. [Accessed: 02-May-2018].

[3] Stop Words, “stop-words,” PyPI. [Online]. Available: https://pypi.org/project/stop-words/. [Accessed: 10-May-2018].

[4] Ólavur Mortensen, "Text Summarization with Gensim", Rare-technologies.com, 2018. [Online]. Available: https://rare-technologies.com/text-summarization-with-gensim/. [Accessed: 27- Mar- 2018].

[5] Scott N. Dobbins. “Re-learning English: Pluralization and Lemmatization,” *Searching for Bole*, 14-Jan-2018. [Online]. Available: https://searchingforbole.com/2018/01/10/lemmatizer/. [Accessed: 02-May-2018].

[6] Radim Rehurek and Petr Sojka, "gensim: topic modelling for humans", Radimrehurek.com, 2018. [Online]. Available: https://radimrehurek.com/gensim/intro.html. [Accessed: 27- Mar- 2018].

[7] Microsoft Developers, “.NET Downloads for Windows,” *Microsoft*. [Online]. Available: https://www.microsoft.com/net/download/windows. [Accessed: 02-May-2018].

[8] MySQL Developers, "MySQL :: MySQL Connector/Net Developer Guide :: 8.2 Entity Framework 6 Support", Dev.mysql.com, 2018. [Online]. Available: https://dev.mysql.com/doc/connector-net/en/connector-net-entityframework60.html. [Accessed: 27- Mar- 2018].

[9] Microsoft Developers, *ASP.NET MVC Overview*. [Online]. Available: https://msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx. [Accessed: 02-May-2018].

[10] Amol Gupta, “Insert/edit link,” *SAP Blogs*. [Online]. Available: https://blogs.sap.com/2016/01/02/an-insight-into-model-view-controller-mvc-in-the-context-of-sap-ui5/. [Accessed: 02-May-2018].

[11] Microsoft Software, ".NET and C# - Get Started in 10 Minutes", Microsoft, 2018. [Online]. Available: https://www.microsoft.com/net/learn/get-started/windows. [Accessed: 27- Mar- 2018].

[12] Scrapy Developers, “Spiders,” *Spiders - Scrapy 1.5.0 documentation*. [Online]. Available: https://docs.scrapy.org/en/latest/topics/spiders.html#topics-spiders. [Accessed: 02-May-2018].

[13] JQuery, jquery.org, *jQuery*. [Online]. Available: https://jquery.com/. [Accessed: 02-May-2018].

[14] Mihalcea, Rada, and Paul Tarau, "Textrank: Bringing order into text." Proceedings of the 2004 conference on empirical methods in natural language processing, 2004.

[15] Blei, David M., Andrew Y. Ng, and Michael I. Jordan, "Latent Dirichlet allocation." Journal of machine Learning research, 993-1022, 2003.

Appendices

Appendix A: Summary Example for Dell Inspiron 13 500 2-in-1

Figure 11: Keyword word cloud for Dell Inspiron 13 500 2-in-1

Figure 12: First Topic word clouds for Dell Inspiron 13 500 2-in-1

Figure 13: Second Topic word clouds for Dell Inspiron 13 500 2-in-1

Figure 14: Third Topic word clouds for Dell Inspiron 13 500 2-in-1

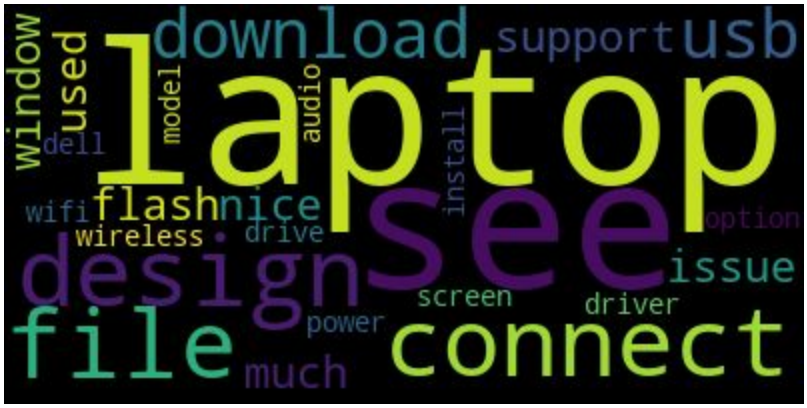

Figure 15: Fourth Topic word clouds for Dell Inspiron 13 500 2-in-1

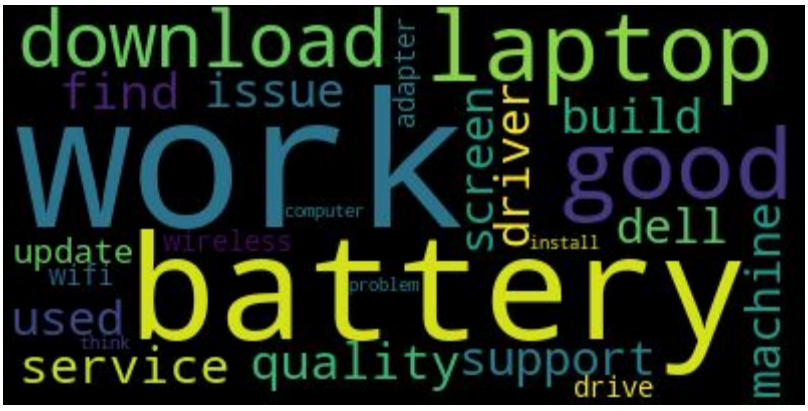

Figure 16: Fifth Topic word clouds for Dell Inspiron 13 500 2-in-1

Keywords for “Dell Inspiron 13 500 2-in-1”
"laptop", "dell", "screen", "work", "working", "driver", "problem", "useful", "use", "good", "time", "wifi issue", "window", "install", "installation", "great", "come", "battery", "buy", "buying", "keyboard", "replace", "replacement", "news", "new"

Text Summary for “Dell Inspiron 13 500 2-in-1”

I've never HATED a single piece of electronics so much in my entire life.\" It is so hard to navigate a page, to use features, even to cut and paste, everything about this laptop is SIMPLY RIDICULOUS!!!) From original Review: I hate the built in apps., the functions don't work, the apps revert with updates, it won't allow you to disable or remove or even change default apps in some cases. INSTALLATION ISSUES WITH DELL DRIVERS DOWNLOADED FROM DELL Anyone downloading driver files from the Dell support site for this laptop should know that as of (4/19/2016) there is a problem with the files. WIndows 10 works beautifully on this laptop, On the flip side I think the product that I have got has some inherent issue with the in-built speakers. Especially the driver under network section with name - Intel PROSet/Wireless 3165 WiFi Driver I downloaded the above driver on a different computer and ported to this new Dell laptop via flash drive.After installing above driver, this product starts connecting to Wifi and then I felt that I can use this laptop. To correct the problem, perform the following steps (assuming your laptop will not stay connected to the internet long enough to download the updated driver): 1. However, Dells very helpful tech synced our Tag Number and fixed the WiFi driver issue and has worked flawless ever since. Not sure if this is Dell's fault or Amazon's fault, but it's looking like I'm not going to be able to use this laptop for a while as they will still need to send me replacement hardware (that hopefully I'm able to install myself?) Ridiculous for a brand new laptop. These new laptops don't come with LAN ports anymore.The wifi worked once the new driver was installed. Not Good, after 2-3 weeks of purchase i decided to drain full battery and recharge it, to see how long will it take to turn back on, well not a good experience, it tool at least an hour and when battery was 60% for it to turn back on, which is terrible, so i try to replace it with Amazon, but representative kept telling me, my address is deactivated and also my card is not on list, but both are same and there was no problem when i purchased it, after 3-4 times of talking they decided to partially refund some amount for battery so i can buy another, problem with that is amazon or any other seller doesnt sell it, and no one knows which battery i need to replace, so i am stuck with that, but other than that it is a good product, but i am disappointed since it should be working nicely Very nice machine! The ONLY reason I am not giving it 5 stars is a slight problem I had with the Intel Graphic Drivers and the fact that I had to run the actual Windows 10 download after receiving it. It's a laptop, it has a touch screen, it works. Having just purchased this computer on 05-21-2016 on sale at a brick-n-mortar big box office store, I think the Dell update tool automatically remedies the problem of the instability with the Intel ProSet wireless adapter.

Appendix B: Database Diagram

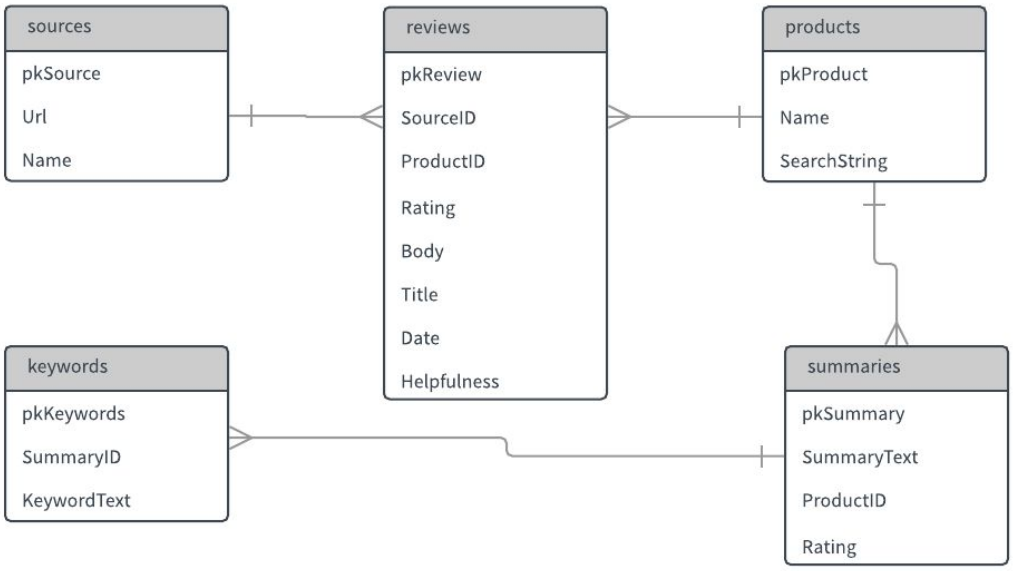

Figure 17: Database Diagram