Rally, a One Stop-Shop for *Reddit* data and Insights

by

Kevin J. Eger

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

B.SC. COMPUTER SCIENCE HONOURS

in

Unit 5

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Okanagan)

April 2016

Abstract

Reddit is *the front page of the internet*, a slogan the company has coined and rightfully lived up to. It is a website which brings together members of all communities in a similar style to a typical forum but with much more structure and a lot more traffic. The open nature of *Reddit* captures over 200 million unique visitors a month. With such traffic screams the demand for data analysis through a human-interpretable medium. Data analysis on *Reddit* has been done before, but this thesis focuses on bringing the data gathered into an easily consumable format. We will explore the implementation and results of querying the *Reddit* API, generating aggregate statistics, querying large data dumps of historic *Reddit* data with *Google BigQuery*, analyzing and labelling the content of *Reddit* using *Google Cloud Vision*'s image recognition, providing an innovative technique for consuming *Reddit* and the use of unsupervised machine learning to draw powerful conclusions. The result is a system called *Rally* which brings together the busy and wild community of *Reddit* through clear and effective data aggregation, inference and visualization.

Table of Contents

List of Tables

List of Figures

Acknowledgements

Work on this thesis was widely facilitated with help from my supervisor, Dr. Ramon Lawrence, through weekly meetings where ideas and progress were discussed extensively. It is also important to acknowledge Dr. Jeff Andrews for his support in advising on machine learning techniques which were implemented as described later. I would also like to acknowledge the *Reddit* community as a whole for their feedback on a portion of this project which was launched for public use.

Chapter 1

Introduction

Reddit is a news and entertainment website whose content is generated by members of the community. Users submit text posts or direct links similar to a typical forum setting. Registered users can vote on submissions, yielding an ordered online bulletin board. Furthermore, what makes *Reddit* unique is that content is subsectioned into different areas of interest called "subreddits". Some of the top subreddits include *movies*, *funny*, *AskReddit*, *food* and *news*. As of March 3rd, 2016 *Reddit* had 231,625,384 unique users a month viewing a total of 7,517,661,034 pages. The company was founded 10 years ago and has quickly become the most central place on the internet to partake in conversation or consume a wide array of content.

For years, data analysis has been used in many industries to give companies and organizations more information in making business decisions and verification of their models and structures. Whether mining huge data sets, looking at specific use cases or aiming to prove or disprove a theory, companies and organizations alike aim to do one thing: identify and discover patterns, relationships and inferences that are not immediately apparent.

A motivator for this thesis was existing technology for *Twitter* insights. The community-content driven nature of *Twitter* parallels that of *Reddit*. There have been many cases of academic research and production level software released for *Twitter* data management, pattern identification/tracking and the existing infrastructure in the *Twitter* space can be largely replicated and modified to suit *Reddit*.

Rally is the name of the implemented suite of tools built. The inspiration for the name is derived from the formal definition of the word: "to come together". *Rally* combines the accessing, processing, aggregating and visualising of *Reddit* data in one central implementation. It is delivered in the form of a web application so as to be accessible by the largest possible

array of people. This thesis will cover the research, experimentation and considerations that ultimately produced the final "product", *Rally*.

Chapter 2

Background

To best understand this thesis and the work done, it is necessary to be introduced to the relevant technologies and key terms which will be heavily referenced and built upon.

2.1 Key Terms and Definitions

- *Reddit*: entertainment, social news network service (website)
- Karma: how much good a user has done for the *Reddit* community quantified by submissions links that people upvote
- *Google Cloud Platform*: Google's core infrastructure, data analytics and machine learning
- API: application program interface
- *CUDA*: parallel computer platform for harnessing the power of an Nvidia GPU (graphics processing unit)
- Subreddit: a *niche* forum on *Reddit* focused around a common topic or content type
- AJAX: web development techniques used on the client-side to create asynchronous requests

2.2 Reddit

2.2.1 History

The company was founded by two new graduates of the *University of Virginia*, Steve Huffman and Alexis Ohanian, in June 2005 [Gua05]. After a couple years of growth, *Reddit*'s traffic exploded and the service went viral. The creators were quick to release *Reddit* Gold, which offered new features and usability improvements, providing the company with a primary source of income.

2.2.2 Community

Reddit thrives on its open nature and diverse content fully generated by the community [Atl14]. The demographics *Reddit* serves allows for a wide range of subject areas thus having the ability for smaller communities to digest their niche content. Subreddits provide a very unique opportunity by raising attention and fostering discussion that may not be seen as mainstream and covered by other news or entertainment mediums.

Reddit as a company and as a community has been known for several philanthropic projects both short and long term. A few notable efforts are:

- Users donated $185,356 to Direct Relief for Haiti after the earthquake that struck the country in January 2010
- *Reddit* donates 10% of it's yearly annual ad revenue to non-profits voted upon by its users [Red14]
- Members from *Reddit* donated over $600,000 to DonorsChoose in support of Stephen Colbert's March to Keep Fear Alive [Don10]

Chapter 3

Technical Stack

Rally is a project that explores many different types of data accessing methods, processing techniques and visualizations. Due to the nature of web applications, it is no surprise that *Rally* is implemented with modular programming as a key focus. Several design choices and system architecture decisions are what will allow this project to be easily continued and built on. The technical stack is broken into components as follows.

3.1 Laravel

Laravel is a PHP web application framework with expressive, elegant syntax [Lar14]. *Laravel* is designed primarily with the motive of removing the repetitive and trivial tasks associated with the construction of a majority of web projects (ie: authentication, routing, sessions, etc.). *Laravel* aims to make the development process a pleasing one for the developer without sacrificing application functionality [Lar14]. The accessible and powerful framework was chosen for its existing familiarity and power to implement a project spanning many domains.

3.1.1 MVC

Laravel follows the traditional Model-View-Controller design pattern. Models interact with the database through the *Eloquent* ORM providing an object oriented handle on information. Controllers handle the requests and retrieve data by leveraging the models. Views render the web pages and are returned to the user.

This intrinsic design pattern was followed tightly alongside the addition of

a repository layer. As discussed later, *Rally* interacts with several external resources such as the *Reddit* API and the *Google Cloud Platform*. These external resources house gigabytes of data, thus storing them locally and accessing them through a model is counterproductive. To retain the structure of the MVC framework, a repository layer is built on top of the models. This allows for the convenience of a seemingly object oriented interaction with data outside of the application. Not only does it allow for convenient method calls but also abstracts logic away from the controllers, leaving them as slim as possible. This is a vital design philosophy to modern web development as it modularizes code to ensure a more rigid flow and testable code-base. Basic examples from *Rally* utilizing each level of the MVC framework as well as the repository layer. An example of model, view, controller and repository use are seen in Figure 3.1, 3.2, 3.3 and 3.4 respectively.

```
$cluster_image = Cluster::where("name", $subreddit)->
    ↪ first();
```

Figure 3.1: Example of retrieving the first *Cluster* model where the name field matches.

```
<select name="labels" . . . multiple="">
  @foreach($labels as $label)
    <option value="{{ $label }}">{{ $label }}</option>
  @endforeach
</select>
```

Figure 3.2: Example demonstrating how objects passed to the view are utilized and iterated over to display the options for the index page of *Content Search*. *Laravel* leverages an HTML templating engine called *Blade* which allows for convenient variable dumping and interaction.

```
public function show(Request $request)
{
  $subreddit = $request->get("subreddit");
  $about = $this->phpraw->aboutSubreddit($subreddit);

  return response()->view("subreddit.show", [
    "subreddit" => $subreddit,
    "about"     => $about->data,
    "tagline"   => "A look at /r/" . $subreddit
  ]);
}
```

Figure 3.3: Example of the show() function in the SubredditController. This method retrieves the necessary data, sends the data to a blade view (subreddit/show.blade.php) and finally returns a rendered instance of that view.

The repository layer is utilized primarily to wrap auxiliary data sources. This gives them a similar feel and interaction as a traditional model. Seen in Figure 3.4, a RedditorRepository instance is injected into the RedditorsController class which is then used in its internal functions to gather data using the phpRaw *Reddit* API wrapper in a chainable method technique identical to a traditional model.

```
protected $redditor;

public function __construct(RedditorRepository
    ↪ $redditor)
{
  $this->redditor = $redditor;
}
...
public function show(Request $request)
{
  $user = $request->redditor;
  $subreddits = $this->redditor->getUserSubmitted($user
     ↪ )->getSubredditsList();
  ...
}
```

Figure 3.4: Code snippets from the RedditorController which leverages the power of a repository layer to make chain-able function calls to an auxiliary data source.

3.2 Storage

Databases used to house the necessary persistent information for the application. A local *MySQL* database and cloud-based *BigQuery* database.

3.2.1 *MySQL*

MySQL is an open-source relational database management system (DBMS). In *Laravel*, it is the default database system largely because of its *plug and play* nature. The *MySQL* database is what saves the caching layer as described in detail throughout the implementation section. A visual representation of the schema can be seen in Figure 3.5.

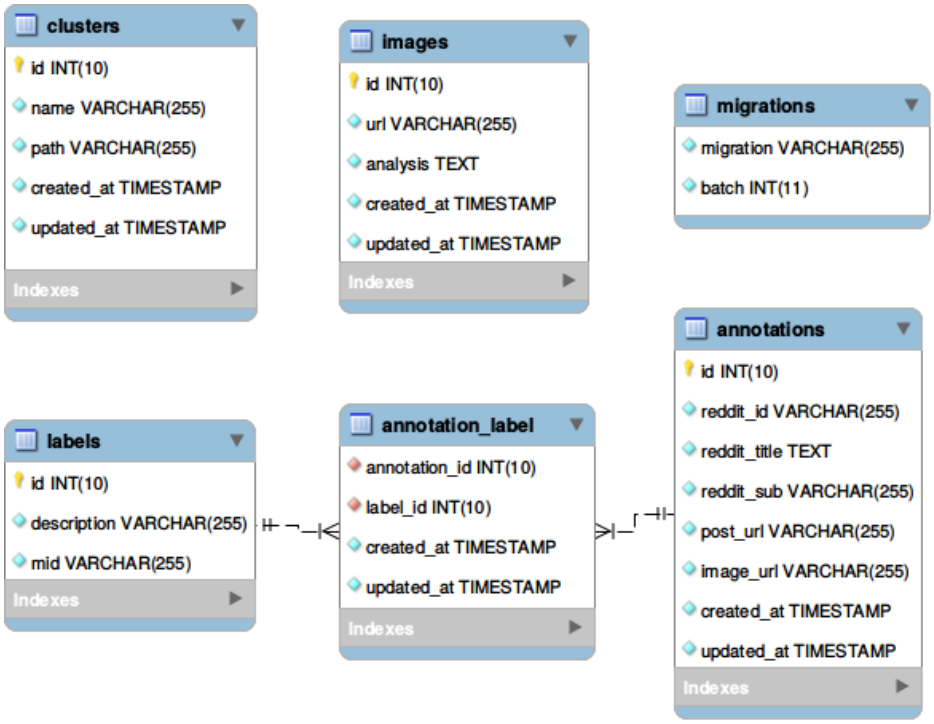


Figure 3.5: An ER diagram representing the *MySQL* database schema.

3.2.2 BigQuery

Querying massive datasets can not only be time consuming but also expensive without the right hardware, infrastructure and software. *Google* alleviates this problem with *BigQuery*, an incredibly fast cloud-based storage platform. It is infrastructure as a service (IaaS) that handles all the hard work of both creating and accessing large data sets. Using the processing power of *Google*, a user can get up and running with *BigQuery* in a matter of minutes. The service can be used via their web UI, command-line tool or the REST API using one of the many client libraries.

In November 2015, user `/u/Stuck_In_the_Matrix` of *Reddit* collected all submission data from 2006 to 2015. He had effectively bundled 200 million submission objects, each with score data, author, title, self_text, media tags and all the other attributes that are normally available via the *Reddit*

API. The dataset complemented the *Reddit* comment corpus he released a couple months prior. When the data was initially made publicly available, he released it as a torrent where developers interested in using it could download their own local copies. Developers were all downloading the data for use either on their local machines or a cloud server. The problem with this is that even with one of the most powerful desktop computers, loading the entire dataset into RAM is not feasible. Search times and joining (cross table) operations were expensive.

Soon after the release of this torrent, one of the lead engineers of *Google BigQuery*, Felipe Hoffa, uploaded the data to *BigQuery* and made the dataset publicly available. Each month, the dataset is updated with the latest information collected from the *Reddit* API.

With the convenience of *BigQuery*, it is now possible to query gigabytes of historic *Reddit* data in a matter of seconds. Listed below are a couple of the integral queries used in *Rally*, their sizes and execution times.

```
SELECT subreddit, total, sub_hour, num_gte_3000
FROM (
  SELECT
    HOUR(SEC_TO_TIMESTAMP(created - 60*60*5)) as
        ↪ sub_hour,
    SUM(score >= 3000) as num_gte_3000,
    SUM(num_gte_3000) OVER(PARTITION BY subreddit)
        ↪ total, subreddit,
  FROM [fh-bigquery:Reddit_posts.full_corpus_201509]
  WHERE YEAR(SEC_TO_TIMESTAMP(created))=2015
  GROUP BY sub_hour, subreddit
  ORDER BY subreddit, sub_hour
)
WHERE total>700
ORDER BY total DESC, sub_hour
```

Figure 3.6: The BigQuery SQL for finding the best hours to post on *Reddit*. This query processes 5.00GB across one table in roughly 8 seconds (roughly 1.5 seconds when cached).

```
SELECT RIGHT('0'+STRING(peak),2)+'-'+subreddit, hour, c
FROM (
  SELECT subreddit, hour, c, MIN(IF(rank=1,hour,null))
  OVER(PARTITION BY subreddit) peak
  FROM (
    SELECT subreddit, HOUR(SEC_TO_TIMESTAMP(created_utc
        ↪ )) hour, COUNT(*) c, ROW_NUMBER()
    OVER(PARTITION BY subreddit ORDER BY c ) rank
    FROM [fh-bigquery:Reddit_comments.2015_08]
    WHERE subreddit IN (%subreddits)
    AND score>2
    GROUP BY 1, 2 )
    )
ORDER BY 1,2
```

Figure 3.7: Viewing activity (number of submissions) on subreddits over time. The wildcard *%subreddits* is replaced with a comma-separated string of subreddits. This query processes 1.49GB across one table in roughly 2.5 seconds (roughly 1.1 seconds when cached).

Facades in Laravel with Google Services

In web programming, quite often developers will need access to static references of classes. Facades provide a static interface to such classes that are available in the application's service container. By default *Laravel* ships with several facades. These static proxies to underlying classes in the service container provide the benefit of a terse, expressive syntax while maintaining more testability and flexibility than traditional static methods.

The facade class itself only needs to implement a single method `getFacadeAccessor()`. It is that method's job to define what to resolve from the container. Behind the scenes, the base facade class (which all facades must extend) makes use of a magic-method, `__callStatic()`, which defers calls from the facade to the resolved object.

```
public function register()
{
  $this->app->bind('google', function () {
    $client = new Google_Client();
    $client->useApplicationDefaultCredentials();
    $client->addScope(Google_Service_Bigquery::BIGQUERY
      ↪ );

    return new GoogleAPI($client);
  });
}
```

Figure 3.8: Registering the *Google* service provider and binding the facade keyword *Google* to it.

The point of registering a facade may at times seem convoluted and unnecessary. It has always been a topic of discussion amongst the PHP world and most often boils down to personal preference and code readability. The facade approach was chosen particularly for *BigQuery* in this project for a few main reasons:

- Expressive syntax without sacrificing testability of code
- Narrow and well defined class responsibility
- Clean constructor injection to automatically connect to *Google Services* and access the *BigQuery* API
- Explicit declaration defines what the class needs and what the class does

3.3 SciPy

SciPy is a Python based ecosystem of open-source software geared towards mathematics, science and engineering. In particular, this project utilizes the NumPy package for array manipulation and processing, the SciPy package for the hierarchical clustering, linkage matrix generation and dendrogram presentation and finally the Matplotlib package for plotting and

displaying the dendrogram. Each of the utilizations of the packages are broken down further in later sections as they are employed.

Chapter 4

Algorithms and Methods

This chapter describes algorithms integral to the key components of *Rally*. The hierarchical clustering of a subreddit and techniques for image classification are described in detail.

4.1 Hierarchical Clustering

When observing an open environment, a powerful metric for how the community is distributed is discovered with clustering. One of the biggest benefits of hierarchical clustering is that you do not need to know the number of clusters in the data set going into the analysis. It is with hierarchical clustering that within a subreddit, we are able to detect sub-communities. Strategies for hierarchical clustering land within two groups: agglomerative and divisive. Agglomerative is a bottom up approach where each observation starts in its own cluster and pairs are merged as you move up the hierarchy. Divisive is a top down approach where all observations begin in a single cluster and are split recursively down throughout the hierarchy.

To best understand the hierarchical clustering process, we will begin by showing the end result in what is known as a dendrogram. A clustering of users amongst the subreddit */r/movies* is shown as a dendrogram in Figure 4.1. The dendrogram is a visualization in the form of a tree that shows the order and distances of merges throughout the hierarchical clustering. It can be understood as snapshots throughout the linkage of observations. On the x-axis are labels representing numbers of samples (if in brackets) or specific samples (without brackets). On the y-axis are the relative distances (using the 'ward' method described later). Beginning at the bottom of the lines (near the labels), the height of the horizontal joining lines tells us the distance at which that labelled group merged with another label or cluster.

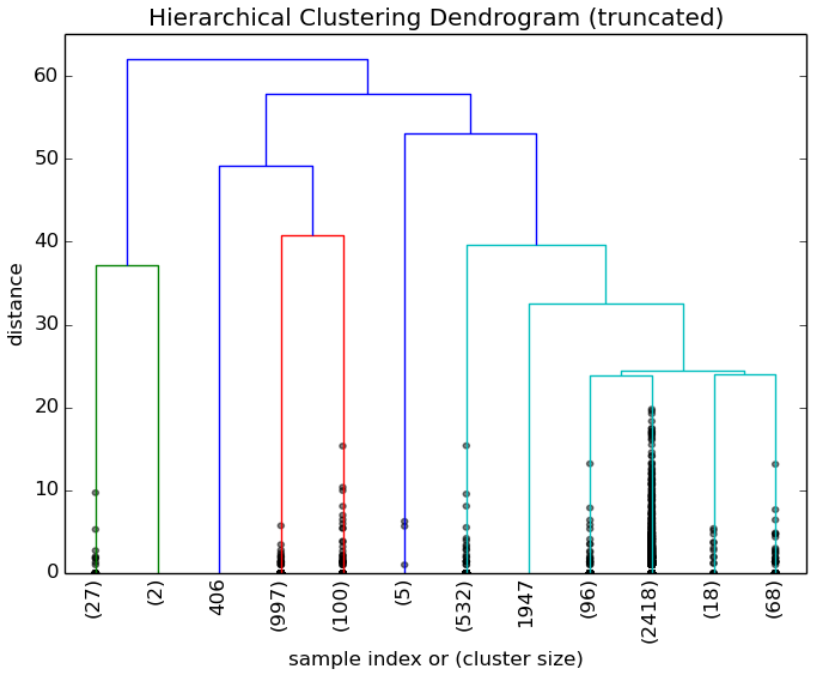


Figure 4.1: A dendrogram representing the hierarchical clustering amongst the subreddit */r/movies*.

For the example shown in Figure 4.1 there are 4265 samples (users) being processed. Shown is a truncated dendrogram, with only the last 12 merges. The small black dots along the vertical lines represent joins that happened prior to the final 12. Truncation is an incredibly useful tool when plotting dendrograms. More often than not, we are only interested in the last few merges amongst the samples. The merge that carries the largest vertical distance will be the merge that attaches the most segregated groups. Again with the example illustrated in Figure 4.1 we see three distinct groups being formed, identified by their green, red and teal colours (left, center and right groups respectively).

Before summarizing the process, here is a concise list of the variables and what they map to:

– X: samples (n*m array), or data points or "singleton clusters"

- n: number of samples
- m: number of features
- Z: cluster linkage array
  - Contains the hierarchical clustering information
- k: number of clusters

4.1.1 The Clustering Process

To begin the clustering, we first gather the necessary data from *Google BigQuery*. The query retrieves the most recent 300 posts for the specified subreddit. A join is then made with the `link_id` from the inner query and a `UNION ALL` with the comment shard tables over the past 3 months. *BigQuery* does not directly support the `UNION ALL` syntax familiar to most sql languages, but instead supports comma separated tables wrapped in a `SELECT *`. After joining up the relations, user accounts that were deleted or made by an auto moderator are filtered out. The remaining authors are grouped by `link_id` and selected out by the number of times they commented on each link. The query as executed in the application can be seen in Figure 4.2. The query processes 9.95GB of data across a total of 4 tables and is completed between 5 and 10 seconds (depending on the subreddit under consideration).

```
SELECT author, link_id, COUNT(link_id) as cnt
FROM (
  SELECT *
  FROM
   [fh-bigquery:Reddit_comments.2016_01],
   [fh-bigquery:Reddit_comments.2015_12],
   [fh-bigquery:Reddit_comments.2015_11]
)
WHERE link_id IN (
  SELECT posts.name
  FROM [fh-bigquery:Reddit_posts.full_corpus_201512] AS
    ↪ posts
  WHERE posts.subreddit = (%subreddits)
  AND posts.num_comments > 0
  ORDER BY posts.created_utc DESC LIMIT 300
)
AND author != '[deleted]'
AND author != 'AutoModerator'
GROUP BY author, link_id
ORDER BY author
```

Figure 4.2: The query executed on *BigQuery* to retrieve all cluster data.

Upon retrieving the data, the X matrix needs to be generated which has n samples and m features. Samples are authors of comments on listings and features are each of the listings. In Figure 4.3 is the algorithm for processing the raw *BigQuery* response into a usable matrix. Because the matrices can become very large in size, we are currently limiting the data gathered by using only the most recent 300 posts. Future work could focus on coming up with a preprocessing technique to predict the anticipated size of response data from *BigQuery* and select an appropriate post number.

```
   input  : raw BigQuery table response
   output: n * m matrix of users and submissions with comment
           frequency values
 1 for each row in response do
       // Save the frequency a user commented on a post
 2     values[author][linkid]= count;
       // Save unique users
 3     if user has not been seen before then
           // Append username to users array
 4         users[]= user;
 5     if link has not been seen before then
           // Append link to links array
 6         links[]= link;
 7 for each user in users do
 8     for each link in links do
           // If a user has commented on a link
 9         if values[user] has array key link then
               // Set [user][link] = count
10             result[user][link]= values[user][link];
11         else
12             result[user][link]= 0;
13 return result;
```

Figure 4.3: Preparing the *BigQuery* response data for clustering.

Upon generating the X matrix, the results are dumped out to a json encoded file. The path to the json file is then passed along with a call to execute the Python script.

Generating the linkage matrix Z in Python with the help of *SciPy* is straightforward. An *(n-1)* by 4 matrix Z is returned. At the i-th iteration, clusters with indices *Z[i, 0]* and *Z[i, 1]* are combined to form cluster $n+i$. A cluster with an index less than n corresponds to one of the n original observations. The distance between clusters *Z[i, 0]* and *Z[i, 1]* is given by *Z[i, 2]*. The fourth value *Z[i, 3]* represents the number of original observations in the newly formed cluster. The algorithm starts with a forest of clusters.

When two clusters s and t from this forest are combined into a single cluster u, s and t are removed from the forest and u is added to the forest. The algorithm is complete when only one cluster remains in the forest and this cluster becomes the root. A distance matrix is maintained at each iteration.

The *d[i,j]* entry corresponds to the distance between cluster i and j in the original forest. At each iteration, the algorithm must update the distance matrix to reflect the distance of the newly formed cluster u with the remaining clusters in the forest.

There are multiple methods for calculating the distance between newly formed clusters u and v. We elect to use the ward method. Suppose there are */u/* original observations *u[0],...,u[/u/-1]* in cluster u and */v/* original objects *v[0],...,v[/v/-1]* in cluster v. Recall s and t are combined to form cluster u. Let v be any remaining cluster in the forest that is not u. Given these definitions for observations and objects, the ward method calculates distance between the newly formed cluster u and v as follows in equation 4.1. Where u is the newly joined cluster consisting of clusters s and t, v is an unused cluster in the forest, $T = /v/ + /s/ + /t/$.

$$\sqrt{\frac{|v|+|s|}{T}d(v,s)^2 + \frac{|v|+|t|}{T}d(v,t)^2 - \frac{|v|}{T}d(s,t)^2} \tag{4.1}$$

The final piece of the puzzle is visualizing the results using a dendrogram as introduced at the beginning of this section in Figure 4.1. The full code used to visualize the linkage matrix is outlined in Figure 4.4. As we can see, by simply specifying title, label, turning parameters and p (the number of final merges to show) produces an intuitive dendrogram with clear colour and stage distinction.

```
plt.title('Hierarchical Clustering Dendrogram (
    ↪ truncated)')
plt.xlabel('sample index or (cluster size)')
plt.ylabel('distance')
plt.gcf().subplots_adjust(bottom=0.15)
dendrogram(
        Z,
        truncate_mode='lastp',  # show only the last p
            ↪ merged clusters
        p=12,  # show only the last p merged clusters
        leaf_rotation=90.,
        leaf_font_size=12.,
        show_contracted=True,  # to get a distribution
            ↪ impression in truncated branches
)
# plt.show()
plt.savefig('/your/file/location/cluster.png')
```

Figure 4.4: Drawing the dendrogram using matplotlib.

Summary of the results of a dendrogram:

- horizontal lines are cluster merges
- vertical lines indicate which clusters/labels were part of merge forming that new cluster
- heights of the horizontal lines express the distance needed to be "bridged" to form the new cluster

It is the distance jumps and gaps in the dendrogram that are of value when interpreting the data. When the jump is large, it indicates that two groups are being merged together that maybe should not be merged. In other words, we have identified two potentially unique groups that form independent clusters.

4.2 Image Classification

Image recognition was employed to effectively automate and scale the labelling of media content submitted to *Reddit*.

Human brains make vision seem very trivial, as it doesn't take much effort for humans to distinguish between a jar of alphagetti and a wasps nest (a seemingly very random example but proved to actually be a difficult task). But these are very hard problems to solve with a computer, they only seem easy because our brains are incredibly good at understanding visual queues.

Over the last few years, the field of machine learning has made tremendous progress on addressing these difficult problems. In particular, deep convolutional neural networks can achieve reasonable performance on hard visual recognition tasks. Often matching or exceeding human performance in some domains [Ten].

To classify images with labels, a first attempt was made using *Google TensorFlow*, the recently open sourced machine learning toolkit by Google. In particular, we focused on implementing and leveraging the power of Inception-V3 [SVI+15] - the newest model for identifying higher level features into classes.

TensorFlow is a very complex API for programmers to use either CPU, GPU or in some cases both (using *CUDA*) devices. The barrier to entry is quite high, but upon learning the flow of data and architecture of the infrastructure is a very powerful tool. We will not go into detail on the implementation as it is not relevant to the underlying use.

Upon implementing the image recognition class using *TensorFlow*, an API was built that allowed for convenient calls to classify images sent along as POST data. This system was optimal as it allowed for independent testing and debugging. The major downfall was the lack of speed with the implementation for analysing the image. To reduce the computation time, *CUDA* was used and the algorithm was altered to run in parallel on a GPU. The main struggle with this implementation was working with the *TensorFlow* API on a GPU that did not support the latest version of *CUDA*, which is the only version *TensorFlow* is currently (as of April 2016) targeting.

When the GPU version of the image classification was finalized and

tested, computation time was cut in half, but still took anywhere between 2 and 6 seconds to analyse a single image. The results of the classification were also dissatisfying as only an accuracy of roughly 60% was achieved. It was difficult not to give into the sunk cost of sticking with an approach that was built over the course of a month however, as discussed in the implementation section it was undoubtedly the correct choice to abandon *TensorFlow*.

Chapter 5

Implementation

An in depth overview of the technical implementation of an API wrapper, *Rally* and *RallySearch*.

5.1 phpRaw

The *Reddit* API has several endpoints. It is through these endpoints where a client can retrieve posts specific to a subreddit, post a comment, moderate their account and all other actions that are normally available through the consumable web interface. For a single use or specific focus, calling the endpoints explicitly with *cURL* (or another client-side URL transfer) works fine but this strategy quickly fails as needs grow. Due to the wide array of endpoint calls utilized, it was necessary to develop an API wrapper that allowed convenient calls to the API. Such a wrapper already existed for Python, Java, C and a few other languages but not PHP.

An open source wrapper was discovered on *GitHub* but was no longer maintained, was not written to comply with the latest API security requirements (OAuth2) and was missing nearly half of the endpoints. By building on the work done on this API wrapper, a successful implementation was built and is what *Rally* utilizes and depends on for direct *Reddit* data access. The *GitHub* repository from the point at which it was forked and built on is linked in the appendix.

Listed below are functions from *phpRaw* to give a feel for the wrapper.

Get the user submitted data.

```
$phpRaw->getUserSubmitted($user, $limit = 25, $after =
    ↪ null);
```

Get the top 10 hottest listings for the specified subreddit, 'funny'.

```
$phpRaw->getHot('funny', 10);
```

phpRaw was modified to serve as a standalone vendor service brought in through Laravel's default dependency manager *Composer*. By extracting the wrapper to a separate module, updating and maintaining the endpoints is simple as they are changed over time. Using the power of composer and package dependencies, by including the declaration as outlined in Figure 5.1, whenever *Composer* is updated it automatically updates to the latest version of *phpRaw*.

```
...
"repositories": [
  {
    "name": "kevin/phpRAW",
    "type": "vcs",
    "url": "https://github.com/kevineger/phpRAW"
  }
],
...
```

Figure 5.1: Requiring *phpRaw* as a dependency for *Rally* in composer.

5.2 Rally

Rally combines user statistics, big data, subreddit analysis and Rally-Search into one convenient location.

5.2.1 User Statistics

On *Reddit*, users have the ability to view their recent activity (comments, submissions and saved content) and link/comment karma scores. *Reddit* serves this information in a similar fashion to how they display submissions on their site. This technique is effective for listing out a history of comments and submissions but it proves ineffective for quickly interpreting account

details. To alleviate this lack of accessibility, upon entering a user name on the "User Stats" page, users can quickly see the following information on a user:

- User Card
  - Username
  - Unique ID
  - How long they have been a user for
  - Gold and mod status
- Activity over time (Submissions vs. Time of Day)
- Submission Data
  - Karma
  - Top submission (with link)
  - Worst submission (with link)
  - Most recent submission (with link)
  - Average karma
- Comment Data
  - Karma
  - Top comment (with link)
  - Worst comment (with link)
  - Total comments (with link)
  - Average karma
- Itemized and labelled list of subreddits posted to with badge counts for frequency and highlighting for top

The user statistics section gathers information exclusively from the API wrapper, *phpRaw*. Calls to the wrapper are made through a repository layer, as described in the technical stack section under the MVC subsection. It is in surrounding the *phpRaw* calls with a repository layer that we can save on API calls and thus reduce the time it takes to gather the necessary information through method chaining. An example call from the controller

to the repository which in turn calls *phpRaw* is seen in Figure 5.2. When `getUserSubmitted($user)` is executed, a large response object consisting of most the necessary information for the entire page is returned. By storing this information on the object and making the method chainable (returns an instance of the object) we can have easy and most importantly expressive syntax in calling the approriate information, for example:
`$this->redditor->getUserSubmitted($user)->getTopUpVotes();`.

```
$subreddits = $this->redditor->getUserSubmitted($user)
    ↪ ->getSubredditsList();
```

Figure 5.2: An example of a call to *phpRaw* through the controller via a repository.

To demonstrate the importance of wrapping the API with *phpRaw* and the ease of calling it, the php code for generating the vector of active hours for the "Activity Chart" can be seen in Figure 5.3.

```
public function activeHours()
{
  $hours = array_fill(0,24,0);
  foreach ($this->getSubmissions() as $submission)
  {
    $time = Carbon::createFromTimestampUTC($submission
      ↪ ->data->created_utc);
    $hour = $time->hour;
    $hours[$hour]++;
  }

  return $hours;
}
```

Figure 5.3: The code to generate the vector of active user hours.

5.2.2 Subreddit Clustering

The information on what hierarchical clustering is and how it was implemented are discussed in detail in Section 4.1. This section will discuss the results from the utilization of clustering and how they are presented to the user.

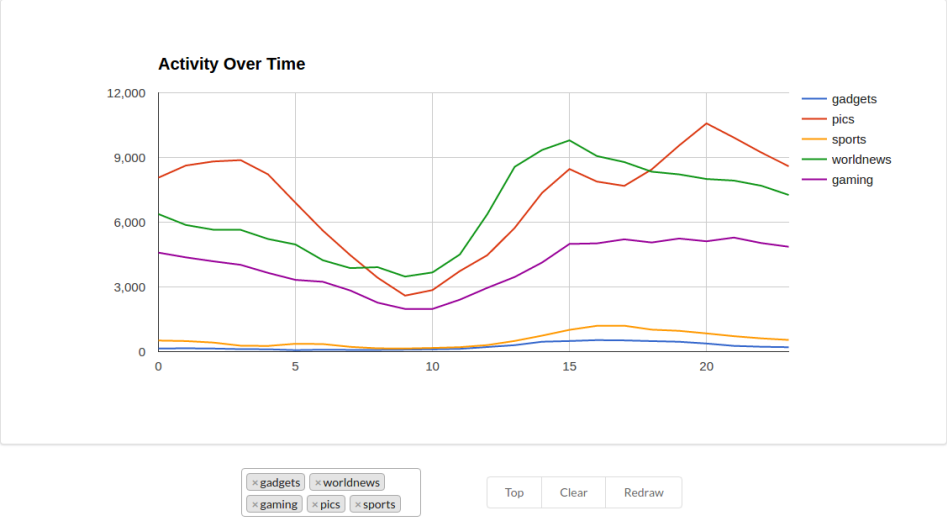
Upon landing on the "Subreddit" page and entering the desired subreddit, two main pieces of information are presented: "Sub Info" - basic information card about the subreddit and "Clustering" - the dendrogram.

The "Sub Info" card contains the subreddit name, unique ID, subreddit logo (if applicable), motto, description and current number of users and subscribers.

The "Clustering" section displays the dendrogram as previously seen in Figure 4.1. Prior to implementing the dendrogram, it was anticipated that sub-communities amongst subreddits would identify and be focused around a subset of conversation topics/themes amongst posts, this was not the case. Upon tracing through some of the dendrograms and examining the specific groups and linkages, it was discovered that "types" of users was in fact the attribute represented by the clustering. In nearly all cases, the largest group clustered together was that of sparse users who comment only a handful of times and infrequently at that. In most other cases there are two well defined groups: those that comment on nearly all of the top submissions and those that comment on just a couple. The results amongst various subreddits have clear distinctions and defining features but tend to follow the same patterns.

5.2.3 Big Data

The Big Data page of *Rally* is intended to harness and demonstrate the power of *Big Query*. From tables spanning sizes of megabytes to tens and twenties of gigabytes, *Big Query* delivers the fastest out of the box relational cloud database.



The first big data snippet on the page is the "Activity Over Time" chart. Users can easily enter the desired subreddits for analysis in the selection box and the graph is redrawn and served to the user in a seamless, AJAX request. As expected, most subreddits take a dip in activity during the nighttime (North American timezones). What is interesting is paring subreddits together that are very similar, for example /r/Programming and /r/ProgrammerHumor. Subreddits with an almost directly equal subscriber list follow a nearly identical activity over time, just with more or less amplitude.

Because *Reddit* is a "reward-based" service (you earn karma on submissions and comments), users often inquire when the best time to post a submission is. This question can be accurately resolved by leveraging the speed with which *Big Query* can read high cardinality tables. By grabbing the highest subscribed subreddits, the results are generated with the query listed in Figure 5.4.

```
SELECT GROUP_CONCAT(STRING(sub_hour)) as hours,
    ↪ subreddit, SUM(num_gte_3000) total
FROM (
  SELECT HOUR(SEC_TO_TIMESTAMP(created - 60*60*5)) as
      ↪ sub_hour, SUM(score >= 3000) as num_gte_3000,
      ↪ subreddit, RANK()
  OVER(PARTITION BY subreddit ORDER BY num_gte_3000
      ↪ DESC) rank,
  FROM [fh-bigquery:Reddit_posts.full_corpus_201509]
  WHERE YEAR(SEC_TO_TIMESTAMP(created))=2015
  GROUP BY sub_hour, subreddit
  HAVING num_gte_3000 > 100
)
WHERE rank<=3
GROUP BY subreddit
ORDER BY total DESC
```

Figure 5.4: Query for getting the best time to post on various subreddits.

Similarly, three other tables exist on the page answering common questions amongst the community. The source code for the queries can be found in `rally/config/constants.php`. Their titles with descriptions are listed here:

- Most popular comments on *Reddit*
  - Rank
  - Count frequency
  - Comment body
  - Average score
  - Count of subreddits comment exists on
  - Count of authors using this comment
  - An example use case of the comment (link to *Reddit*)
- Difference in Cohorts (Account Creation Date)
  - Year account was created

  - Number of users
  - Average number of comments
  - Number of users still presently active
  - Sum of their score
  - Number of gilded users
  - Average body length of comments
- Number of comments by day of the week
  - Day of the week
  - Number of comments

To demonstrate the breadth of possibility in analysing community-based services like *Reddit*, a query and visualisation of the U.S. election candidate mentions was generated. A screenshot of the generated graph can be seen in Figure 5.5 and the query utilized in Figure 5.6.

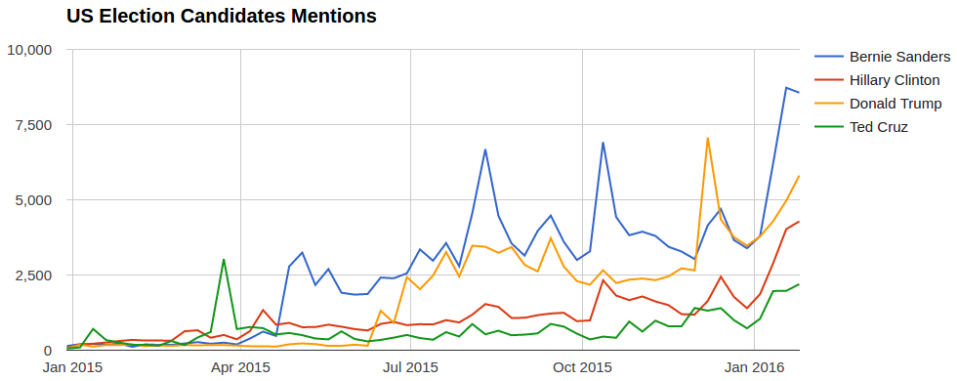


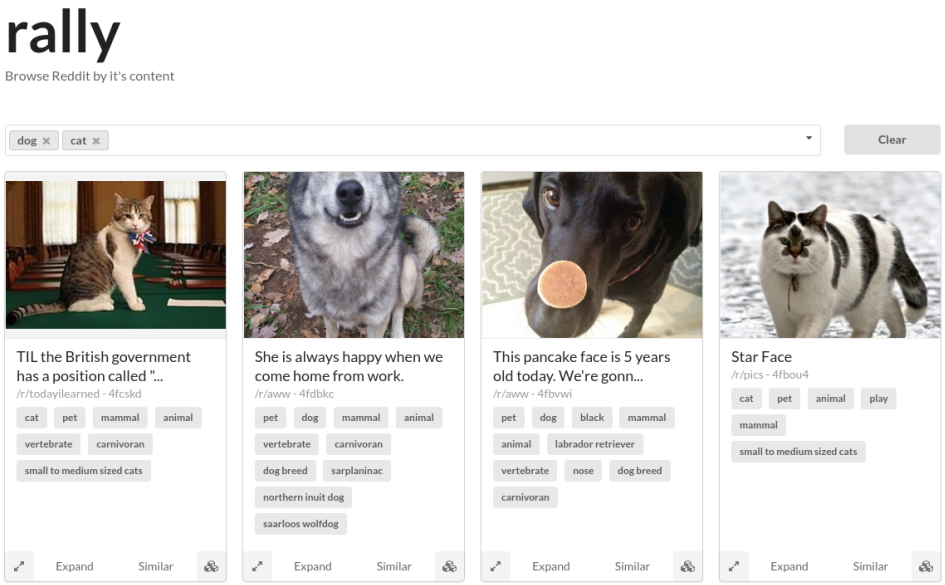
Figure 5.5: U.S. elections candidate mention frequency screenshot.

```
SELECT DATE(USEC_TO_TIMESTAMP(UTC_USEC_TO_WEEK(
    ↪ created_utc*1000*1000,1))) week, SUM(body
    ↪ CONTAINS "Bernie Sanders") BernieSanders,SUM(body
    ↪  CONTAINS "Hillary Clinton") HillaryClinton,SUM(
    ↪ body CONTAINS "Donald Trump") DonaldTrump,SUM(
    ↪ body CONTAINS "Ted Cruz") TedCruz
FROM TABLE_QUERY([fh-bigquery:Reddit_comments], '
    ↪ REGEXP_MATCH(table_id, "201._..$")')
GROUP BY 1
ORDER BY 1
```

Figure 5.6: Query for getting the frequency of top U.S. candidate mentions.

5.3 RallySearch

RallySearch is an alternative way of consuming *Reddit*. Users of *Reddit* are very limited by the existing search functionality. Though there exists a search bar, it is incredibly ineffective and displays close to no results as intended. *Reddit* developers have expressed interest in improving the functionality but have not disclosed any immediate plans to do so. This portion of the thesis aims to offer an alternative to browsing *Reddit* by its content in a visually pleasing and simple fashion. The name of this standalone service has been dubbed *RallySearch* and as discussed later has been launched and received by the *Reddit* community.



5.3.1 Overview

The overall goal of *RallySearch* is to provide users with the opportunity to search *Reddit* by its content - the physically linked images, videos, gifs and articles. *Reddit*'s existing flow for browsing content does not give users the ability to search site-wide for all posts consisting of a specific object. For example, if a user wanted to view all posts pertaining to dogs and cats, they would have to manually search through all the subreddits. The aforementioned may or may not contain the desired content and the only option they would have left is to perform a *Google* search which is even more inefficient and guarantees no degree of accuracy. Using *RallySearch*, labels can easily be specified using the search bar at the top and the page instantly loads all existing classified posts.

When a user first lands on the site, they are presented a page already populated with cards which represent posts. A card consists of the post preview image, title, labels for the image (from image recognition) as well as a few other details and options for navigation discussed later in the UI section.

In summary, top media posts on *Reddit* are sent through *Google Cloud Vision* (Image recognition API) and results are cached. Upon giving each image (or preview in the case of videos and gifs) labels, users can easily view similar content by selecting the desired tag(s).

5.3.2 User Interface

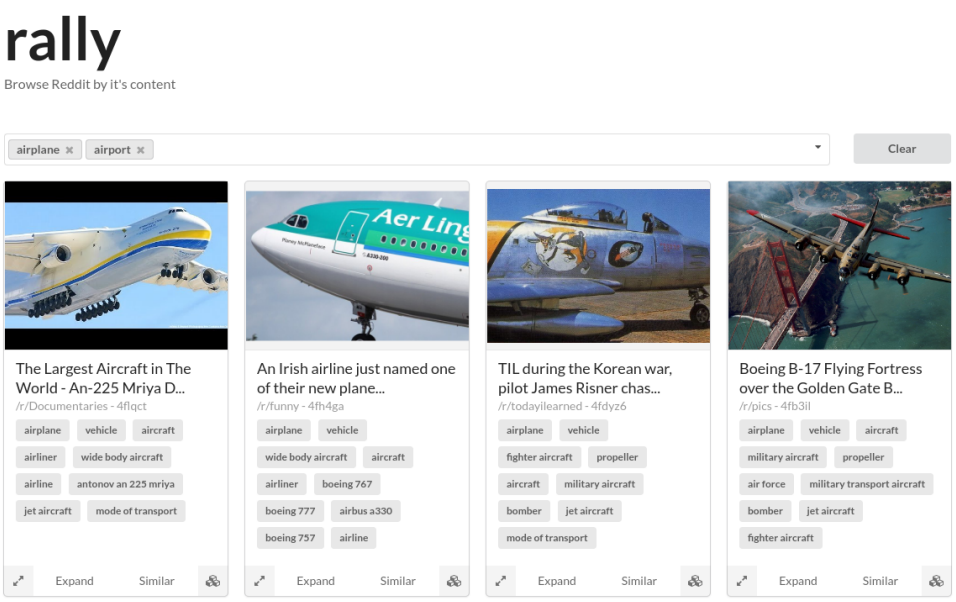
A good user interface (UI) can make or break the success of a website. Too complicated or busy and the site could be left collecting dust on the internet shelves. With *Rally*, the UI balances a clean look with the necessary interactivity coupled with ease of use and simple navigation. The standalone portion of the site *RallySearch* will be surveyed and the design choices that went into designing it as it is slightly more polished than Rally but represents all the same key features.

Semantic UI

Semantic UI is a framework designed for site theming. Key features include predifined CSS for elements, variable tuning, inheritance and responsiveness. Semantic is free, open sourced and MIT licensed. It allows developers to "build beautiful websites fast", with concise HTML, intuitive javascript and simplified expressive CSS class naming.

RallySearch

Near all the interaction on the site takes place on the main index page of the site. Here, users can browse the content and filter by tags if they so wish to.



When a user lands on the index page, the most recent annotations are displayed and are paginated in groups of 20. Splitting the pages is necessary as to avoid loading thousands of models from the database and to avoid screen clutter and "jankiness" (a web design term to describe when a screen stutters while loading dynamic content added with *JQuery*).

Each of the annotations are divided up into cards. A card displays content in a manner similar to a playing card. There is the preview image which is loaded externally from *Reddit* to reduce server load and increase performance, the listing title which is trimmed with ellipsis, the subreddit the image was posted to, all labels given to the annotation and the card functionality.

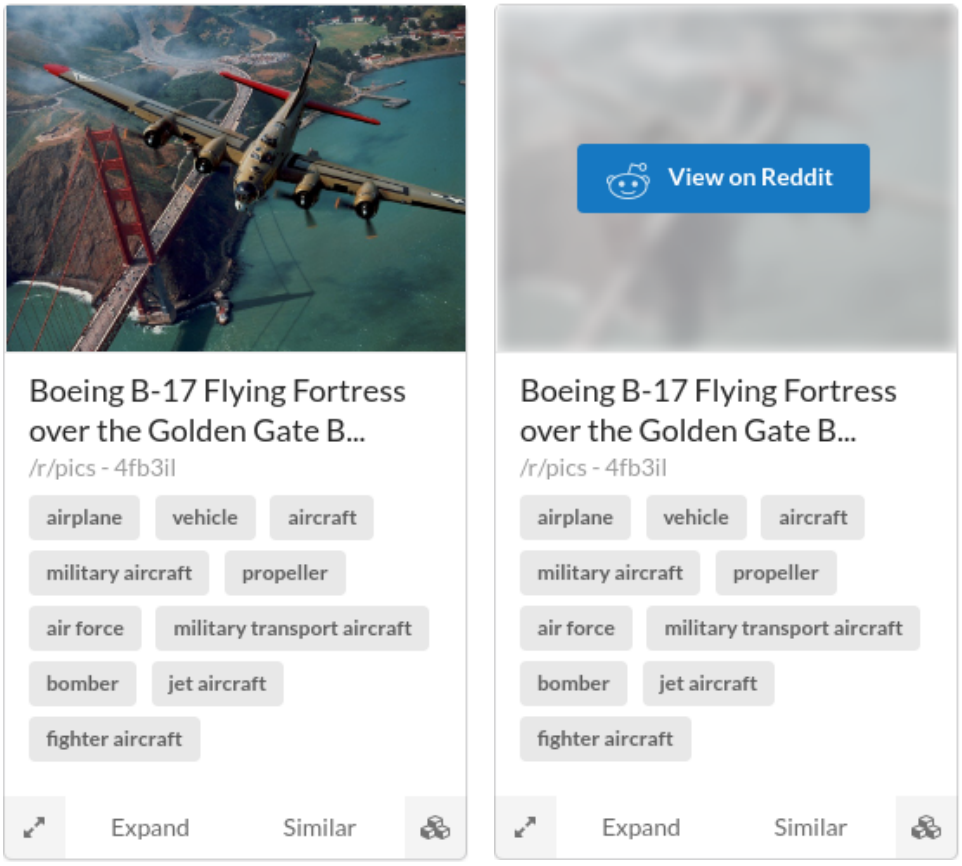


Figure 5.7: A single card without and with hover.

A single card has a few aspects of functionality associated with it. First and foremost is the ability to follow the link through to *Reddit*. RallySearch was designed to simply be a more convenient gateway to *Reddit* content, thus by hovering and clicking "View on *Reddit*" or directly clicking the submission title, the user is redirected to the submission.

If a user wishes to view a larger version of the image, the full title and label list of a card, they simply have to select the "Expand" button in the bottom left. Modals display content in a way that temporarily blocks interactions with the main view of the site, an effective way for ensuring the user is focusing on the desired content and removed from the other "distractions" on the site.

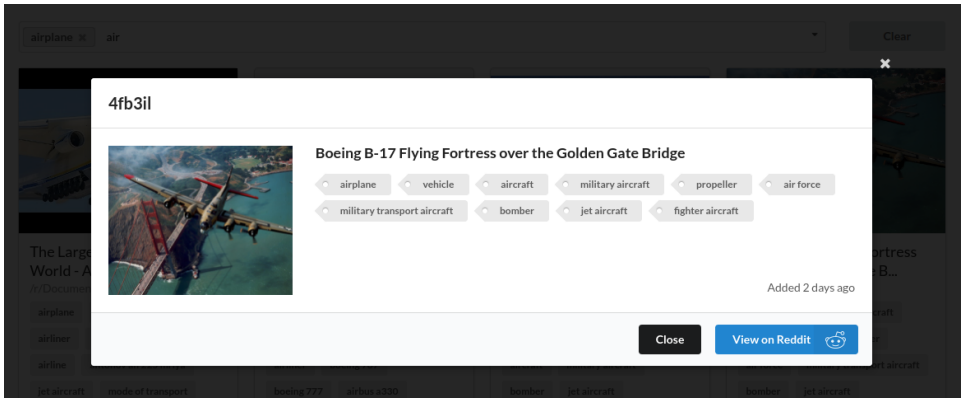


Figure 5.8: A card's modal.

Since the purpose of the site is to browse *Reddit* by labels (content), it makes sense there are a few approaches to do so. First and most clear is the ability to type into the dropdown. As the user enters text, their search is refined from the list of pre-existing labels. There is no limit to the number of labels a user can enter however they must select from the pre-existing labels as all results are loaded from already classified annotations. Users also have the option to select the "Similar" button on a card which instantly loads the corresponding tags into the dropdown. The final way a user can refine their search is by directly clicking on a label in a card. If a user wishes to restart their search, they are free to click the clear button located to the right of the dropdown.

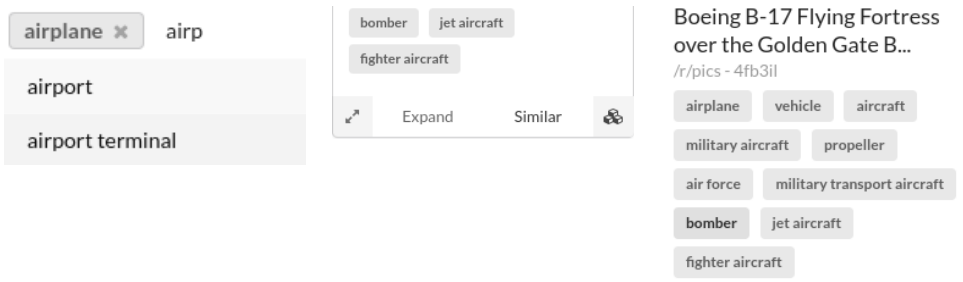


Figure 5.9: A single card without and with hover.

To make browsing the site as fluid as possible, AJAX get requests are

made to the server whenever new content is needed. Whether the user is advancing pages, adding labels or clearing them, the site fetches only the new necessary information and swaps it out with the current. This experience is preferable as it removes the "jankiness" with traditional websites when switching between pages. A single-page application (SPA) is preferable as the fluid experience, similar to a desktop application, does not interrupt the user's actions on the site associated with switching between pages. The dynamically loaded content is also optimized to be as slim as possible, loading quicker than any possible full page reload.

RallySearch is equipped with a very slim and stylish menu bar. The bar is hidden by default but can be activated by click the menu tab always located in the top left of the screen. The tab when not active is an icon of three lines which has become the universal standard for representing context menus that all users are used to interacting with. When the user hovers the icon, it dynamically expands reading "Menu". If the user then clicks on the tab, the menu rolls in from the left pushing the site content slightly the the right in a fluid and smooth fashion. The main content is then dimmed similar to when the modal is displayed as to focus the user on the action they are undergoing.

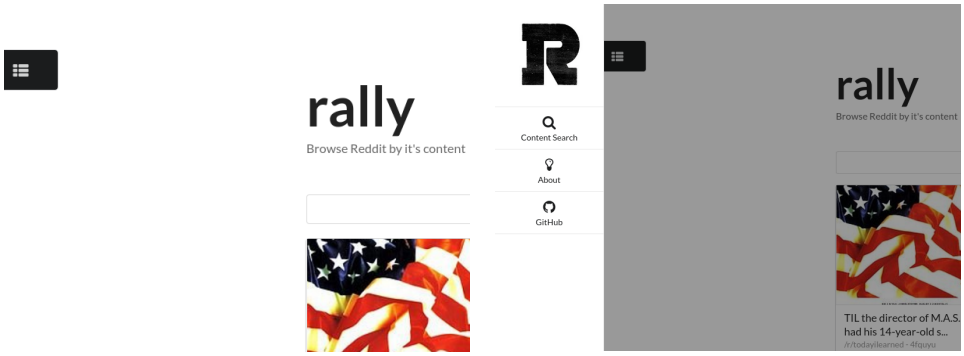


Figure 5.10: The site menu closed and open.

5.3.3 Technical Overview

RallySearch is a slim and highly maintainable web project. It uses two external services, phpRaw (*Reddit* API wrapper created for this project dis-

cussed previously) and *Google Cloud Vision*. The technical workings of *RallySearch* can be divided into two main domains: building the service and consuming the service.

Building the Service

The Google Cloud Vision API enables developers to understand the content of an image by powerful machine learning models in an easy to use REST API. Images are quickly classified into thousands of categories. Individual objects and faces can be detected within an image. The broad set of objects in images are categorized and help improve the Vision API over time as new concepts are introduced and accuracy is improved. The key feature of the API that *RallySearch* uses is label detection. With it, broad sets of categories are detected within an image ranging from dog breeds to wedding dresses.

RallySearch is written with full modularity in mind. This is important because APIs are always subject to change over time and it incorporates two (*phpRaw* and *Cloud Vision*). To accomplish this, a Laravel Job is written which runs on a schedule. The job is executed hourly and at a high level, does the following:

- Creates a new collection
  - *Laravel* iterable object that wraps PHP array
- Retrieves the *hottest* 100 posts on *Reddit*
- Retrieves the *top* 100 posts on *Reddit*
  - Daily
  - Weekly
  - Monthly
  - Yearly
  - All time
- Creates Annotation objects for each of the new listings
  - Checks against database if record already exists

    - Stores unique *Reddit* id, post url, image url (preview), post title, subreddit post was submitted to

- Labels each of the Annotations

    - Sets labelling parameters (feature type: label and max results: 10
    - Builds batch annotation image request object
    - Downloads the image to be analyzed and encodes in base 64 to be sent with the request(s)
    - Sends request and saves response
    - Saves labels for each image, creating new Label objects when needed and reusing existing ones when applicable (An Annotation has many Labels and a Label belongs to many Annotations

The code for this process is slightly too long to include in the thesis but is easily accessible in the repository listed in the bibliography.

Consuming the Service

The main motivation for creating *RallySearch* was to offer a convenient and easy to use way of browsing *Reddit*. Keying in on this motivation it was crucial users were provided with an interface that was easy to adopt and was non intrusive. More on the details of the interface can be found in the UI section, here we will go over a typical users use-case of the site.

The service itself only exists on one page: the content page. Users can browse all labelled posts by scrolling through cards and advancing through pages. For a more refined experience a user can enter a single or multiple tags in the search bar at the top. The bar allows users to only select from a predefined list (labels which exist in the database). Upon clicking or selecting a label(s), the site loads up the appropriate content.

Users are free to navigate their desired content, expand the preview for a larger view of the media and select similar tags by manually selecting those on a post or by using the "Similar" button.

Deployment

RallySearch is deployed on *DigitalOcean*, a cloud computing hosting company. The web app runs on their cheapest tier server (Ubuntu 14.04 with 512MB of memory, 20GB disk space) and performs beyond the needed level. Due to the efficiency and optimization in the web app, it is estimated that this server configuration could serve more than twice the highest reached concurrent users (approximately 50) before suffering any performance costs.

RallySearch has continuous deployment through a third party service called *Codeship*. Each time changes are pushed to the master branch of the repository, *Codeship* is triggered by a GIT hook which brings in the changes, updates the dependencies through composer, dumps all autoloaded and caches and seamlessly brings the server up to date as expected. *Codeship* first brings a virtual server to life with the newest code then copies the changes over to the production code ensuring a near-zero down time. *RallySearch* ensures only code passing the specified automated tests is deployed to production and notifies immediately if any push does not succeed.

5.3.4 Public Reception

Since *RallySearch* is a service for the community and arguably *built by the community* (all the content is theirs), there was a strong desire to push the service live and generate feedback.

RallySearch was made publicly available and posted to a few subreddits intended for developers and enthusiasts to give feedback and test the service. Subreddits the site was posted to include: */r/Frontend* - a subreddit for front end web developers who want to move the web forward or want to learn how, */r/UsefulWebsites* - a compilation of useful websites and */r/design_critiques* - a subreddit to receive critiques on a design. Over the course of just one day, submissions on */r/Frontend* and */r/UsefulWebsites* made it to the top submission of both subreddits and are amongst some of the top posts there over the last year. Though these subreddits are not heavily trafficked, it was a noteworthy accomplishment due to the highly opinionated, dogmatic and domineering nature of the respectful subreddits.

Rather than listing all of the comments, below is a list of a few of the top comments (most upvoted) and most helpful comments that were reflected

with changes and improvements.

- "To be honest the accuracy of the tags blows my mind" - /u/r_park
- "If I'm scrolled down the page, selecting the "similar" button appears to do nothing if it doesn't happen to populate more content. It also seems unintuitive that it adds more tags to my existing tags. This interaction is overall a good idea but a little unclear in terms of what it actually does." - /u/wayspurrchen
- "The content hierarchy for the videos doesn't make a lot of sense/isn't very useful: http://i.imgur.com/RWdyvDL.png As a casual user, I'm unlikely to ever care about the permalink, and I'm most likely to care about the title. The title should also be clickable to take me to *Reddit*. I swapped things around a bit: http://i.imgur.com/1y451Q7.png You could even put something else there like the score for that subreddit, number of comments, etc." - /u/wayspurrchen
- "Overall this is a really killer app. I can't wait to see you polish it! :)" - /u/wayspurrchen
- "You are my new hero." - /u/ChaosElephant
- "This is great - looks nice as well. Awesome job! My only critique at the moment would be to have the full title reveal in some sort of tooltip or something, even just as a title attribute tooltip. At first I didn't notice the expand, or you may just think it expands the photo. It would be nice to see the full title without clicking." - /u/hidanielle

It was great to get the service out there for a non-subjective set of eyes to critique, advise and shape it. Beyond having other *Reddit* users test the site, friends and family were also good help in offering advice and feedback from a new user (unfamiliar with *Reddit*) perspective. Though proper and documented user testing was not conducted, key takeaways with this group include:

- Hooking up links that were expected to direct somewhere but did not initially
- Reorganizing the card structure
- Altering the navigation menu for a more responsive feel

5.3.5 Analytics

Google Analytics were implemented and utilized upon launch. This allowed for the understanding of consumer behaviour, gathering insights, tracking usage numbers and analysing performance of the site. Data tidbits from the analytics are outlined here:

- 736 unique visitors in the first 24 hours
- 0:56 seconds average session duration

Browser	Operating System	Users
Chrome	Android	278
Chrome	Windows	205
Safari (in-app)	iOS	134
Chrome	Macintosh	127
Safari	iOS	77
Safari	Macintosh	35
Chrome	Linux	34
Firefox	Windows	28
Firefox	Linux	19
Firefox	Macintosh	13
Chrome	iOS	12
Edge	Windows	6
Chrome	Chrome OS	4

Table 5.1: Operating system and browser of users.

Country	Users
United States	472
United Kingdom	97
Canada	82
Germany	36
Australia	28
Sweden	25
Netherlands	23
Denmark	20
France	11
Brazil	10
India	10
New Zealand	10

Table 5.2: Countries of users accessing the site.

5.3.6 Open Source

RallySearch is fully open source. The source code is available publicly on GitHub. There has already been a bit of interest with members of the community and the repository has had three stars from developers. The software is MIT licensed.

5.3.7 Moving Forward

The initial launch of *RallySearch* has shown the potential to move forward as an individual project. Though there is no intention to have users strictly use the service and abandon browsing *Reddit* directly, the general consensus has been that it is convenient to relax and browse specific content tailored to a user's interests. Before launching the project and releasing it to the general public by advertising on a higher traffic subreddit, a few improvements and changes need to be made:

- Automated database backups
- Remove the small but noticeable "jankiness" associated with the AJAX page reloading

- Formal user testing
- Build tests
- Update from MySQL to a higher performing database as cardinality increases

Chapter 6

Conclusions

Reddit is forever evolving. The content that is posted, the way the community interacts and the technology itself is always subject to change. Most fields developing tools fear change, as it requires adapting legacy systems or techniques. This project was implemented with that in mind, right from the beginning. All techniques for gathering data, processing and visualizing were built to scale and adapt to change. Though it is true that down the line if a big modification were to happen (ie: the API no longer serves integral data), certain pieces of the software would have to be rewritten. But by and large, the service is solid and built to last.

Having said that, this thesis and project is just the *tip of the iceberg*. The fun part about data analysis is there's no such thing as "done". As new techniques and inspirations arise there is always more to add and infer. Though sometimes an analyst's job is to perform a one-off implementation to retrieve a result, the most fun and complex of problems are those that when solved bring about new questions. This style of analysis is greatly fostered in the *Reddit* space.

Rally has served as a great proof of concept. Combining multiple 3rd party services can be a nightmare and often requires several rewrites. Having completed a first implementation of the service, it brings great satisfaction claiming that an innovative resource for observing *Reddit* inferences has sprung.

There is always room for improvement, both technically and conceptually. Given the current design of *Rally*'s big-data, subreddit and user statistics pages, releasing the site for general public use would require some optimization and refactoring (especially for caching pre-fetched results). *RallySearch* is a web application recently released into the wild of the internet and will undoubtedly require bug fixes as they come up and new features as they are

requested.

More information and current development on *RallySearch* can be found on the *GitHub* repository (https://github.com/kevineger/rallysearch). Though there is currently no intention to build out *Rally* beyond what it has become, the repository will remain uploaded for anyone who wishes to browse for inspiration or potentially build from.

Bibliography

[Atl14] Ama: How a weird internet thing became a mainstream delight, 2014 [cited March 3, 2016]. → pages 4

[Don10] Welcome redditors!, 2010 [cited March 3, 2016]. → pages 4

[Gua05] A new website makes it easier to sift the mountains of news content online - and learns what you like, 2005 [cited March 3, 2016]. → pages 4

[Lar14] Laravel documentation, 2014 [cited March 13, 2016]. → pages 5

[Red14] Decimating our ads revenue, 2014 [cited March 3, 2016]. → pages 4

[SVI+15] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015. → pages 21

[Ten] Image recognition [cited April 18, 2016]. → pages 21