Honours Computer Science Thesis

rally, a one stop-shop for all reddit data

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 generates a large amount of traffic, averaging over 200 million unique visitors a month. With such traffic screams the demand for data analysis through a human-interpretable medium which this thesis covers. Data analysis on reddit has been done before however this thesis focuses on bringing the data gathered in to a easily consumable format. Techniques for consuming less apparent analysis and alternative browsing techniques are covered. 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 and the use of unsupervised machine learning to draw powerful conclusions.

Table of Contents

List of Tables

List of Figures

Acknowledgements

Work on this thesis was widely facilitated with help from 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.

Chapter 1

Introduction

High level overview and motivation for developing this thesis.

1.1 Reddit

Reddit is a a news and entertainment website whose content is sustained by members of the community. Users submit text posts or direct links similar to a typical forum setting. Registered users can vote on submissions bringing order to the posts which yields 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.

1.2 Motivation

For years data analytics has been used in many industries to give companies and organizations better business decisions and verification of their models and structures. Whether they are 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.

An early motivator for this thesis was some existing technology for Twit-

ter insights. The community-content driven nature of Twitter parallels that of Reddit. There has already been a lot of academic research and production level software released for Twitter data management, pattern identification and tracking. The existing infrastructure in the Twitter space can be largely replicated and modified to suit Reddit, an effort which this thesis focuses on starting.

Chapter 2

Background

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

2.1 Terms and Definitions

TODO

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 of notable efforts are as follows:

- 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 access, processing techniques and display forms. Due to the nature of web applications, it is no surprise that *Rally* is implemented with modular programming in mind. Several key components outlined below are what will allow this project to be easily continued and built on. The technical stack is broken in to 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 often painful part of building trivial common tasks to 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 it's 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 retrieving 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 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 are as follows:

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

Figure 3.1: A basic 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: A basic 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.

As mentioned above, 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 in to 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.

```
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: A basic example of the show() functions in the subreddit controller. This method retrieves the necessary data, then sends the data to a blade view (subreddit/show.blade.php) and returns a rendered instance of that view.

```
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 Redditor Controller 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.

3.2.1 MySQL

MySQL is an open-source relational database management system (DBMS). In Laravel, it is the default DBMS largely because of it's *plug and play* nature. The MySQL database is what houses the caching layer as described in detail in the [insert section] section. A visual representation of the schema is depicted as follows:

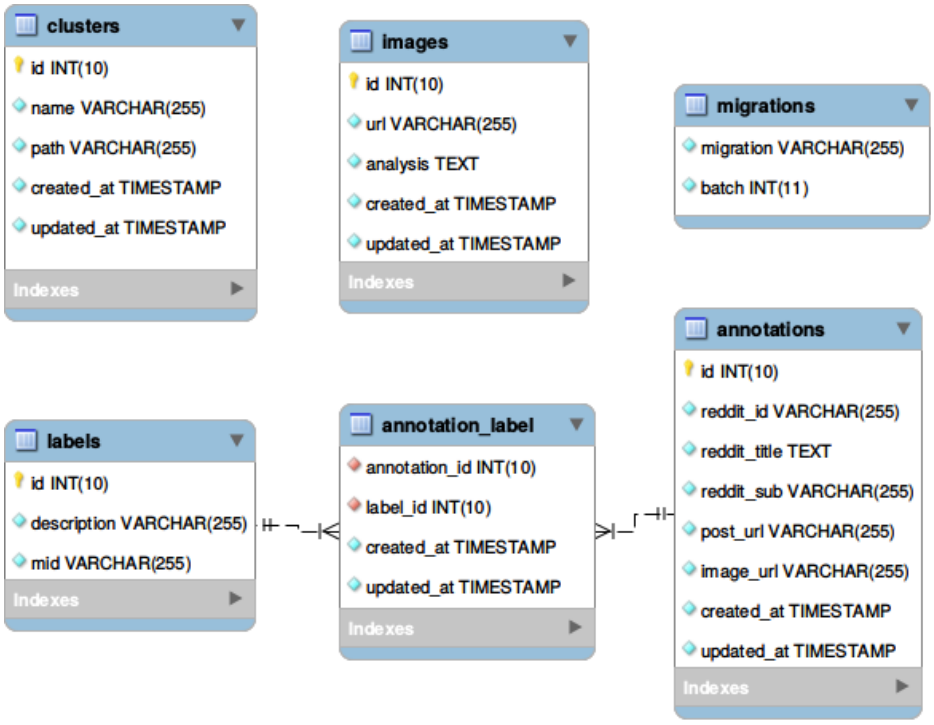

Figure 3.5: An ER diagram representing the MySQL database schema.

3.2.2 BigQuery

Querying massive datasets can not only be time consuming but 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.

Five months ago, user /u/Stuck_In_the_Matrix of reddit collected all Reddit 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 even with one of the most powerful desktop computers, loading the entire dataset into RAM was not feasible. Search times and joining (cross table) operations were expensive.

Conveniently 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 history Reddit data in a matter of seconds. Listed below are a few of the integral queries used in *Rally*, their sizes and the execution time.

```
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 (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 string comma-separated list of subreddits. This query processes 1.49GB across one table in roughly 2.5 seconds (1.1 seconds when cached)

Facades in Laravel with Google Services

In web programming, quite often developers will need access to static references to 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 keyworld *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 a lot of the time boils down to personal preference and code readability. The facade approach was chosen particularly for *BigQuery* part of the project for a few main reasons:

- Expressive syntax without sacrificing testability of code
- Keep class responsibility narrow and well defined
- 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 Python

3.3.1 Scipy

[Insert chunk about Scipy]

3.3.2 Matplotlib

[Insert chunk about Matplotlib]

Chapter 4

Algorithms and Methods

The sets of operations integral to the key components of *Rally*.

4.1 Hierarchical Clustering

When observing an open environment, an equally interesting and powerful metric for how the community is distributed is with clustering. One of the biggest benefits of hierarchical clustering is that you don't need to already know the number of clusters in the data set. 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 it's 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 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 about the distance at which that labelled group merged with another label or cluster.

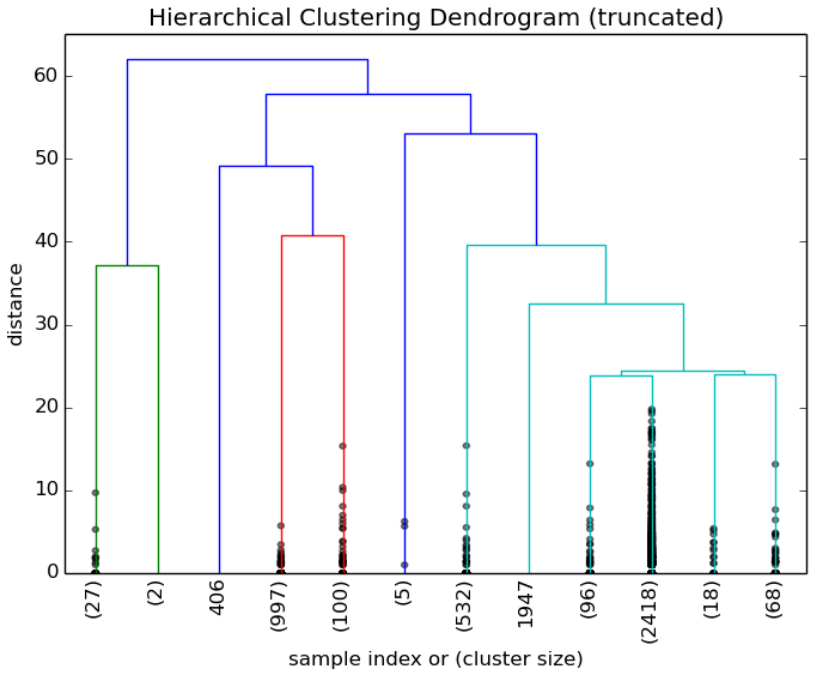

Figure 4.1: A dendrogram representing the hierarchical clustering amongst the subreddit */r/movies*.

For the example shown in Figure 4.1 there are 4265 samples (users) being processed. What is actually being shown is a truncated dendrogram, showing only the last 12 merges. The small black dots along the vertical lines represent joins that happened prior the 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 in Figure 4.1 we can see three distinct groups being formed, identified by their green, red and teal colours.

Before we start summarizing the process, it is sometimes easy to forget what variables map to. A list is provided here:

– 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 the `link_id` and selected out by the number of times they commented on each link. The query as it is 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. As a reminder, in the matrix there are n samples and m features. Our samples are authors of comments on listings and our features are each of the listings. Outlined in figure 4.3 is the algorithm for processing the raw *BigQuery* response in to 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 very 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 in to 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 vizualizing the results using a dendrogram as introduced at the beginning of this section in Figure 4.1. As homage to the simplicity the Python package matplotlib makes this process, the full code used to vizualize 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), we produce 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.

To summarize the results of the seemingly rigorous process outlined above, a brief list follows:

- horizontal lines are cluster merges
- vertical lines tell you which clusters/labels were part of merge forming that new cluster
- heights of the horizontal lines tell you about the distance that 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 shouldn't be merged. Or in other words: we have identified two potentially unique groups that form independent clusters.

4.2 Image Classification

As discussed in the implementation chapter, image recognition was employed to effectively automate and scale the labelling of media content submitted to Reddit.

Human brains make vision seem very trivial, 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 and understanding visual queues.

In 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 in to 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 and modular 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 was 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 it still took anywhere between 2 and 6 seconds to analyse a single image. The results of the classification were also dissatisfying as it only achieved an accuracy of roughly 60%. It was difficult not to give in to 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

Turning the technology into a plan and executing.

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 allows 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. 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 a 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 a specified subreddit.

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

phpRaw was then 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 RallySearch

this is rallysearch

Bibliography

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

[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 3

[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