Predictive Algorithm for Measuring Social Media Activity for NewsWhip

Jack Toner

March 2014
NEWSWHIPP
Predictive Algorithm for Measuring Social Media Activity

24th March 2014

Prepared by: Jack Toner
Supervisor: Brett Houlding
ABSTRACT

The goal for this project was to analyse the nature of trending news media on social networking sites Facebook and Twitter. The aim was to establish the nature of how an article becomes viral and use this information to attempt to develop a predictive model that could determine, from early indicators, the viral potential of any given article. By examining articles tracked by NewsWhip, future trending articles can be recommended to clients.

The model proposed is based on a test articles average dissimilarity to known viral articles. This dissimilarity is based on the Dynamic Time Warped (DTW) distance between the article and a set of training viral articles. This method was found to be slightly less accurate and far more computationally intensive than the existing model.
PREFACE

NewsWhip is a new company on the cutting edge of social media tracking and online journalism. They find the articles trending on social media and relay those trends to publishers worldwide. Their clients include The Guardian, ABC News, The Huffington Post and News Corp. NewsWhip commissioned this project to explore potential new methods of discovery trending news. The application of predictive analytics to their algorithm has the potential to put their clients ahead of the curve when it comes to breaking news and trends.

While the algorithm proposed was an effective tool for article discovery, it failed to surpass the performance of the existing model. However, the project has succeeded in exploring potential alternatives for article classification, and has value in proving that the model utilized by NewsWhip is the most effective currently available.

The main challenge faced was the volatile nature of how journalism and social media interact. Trying to predict the public response to a story is incredibly complex, but considering the disruptive influence that social media and the Internet have had on traditional journalism, I believe this field of study will continue to grow at a rapid rate and have an exciting future.

I would like to thank Brett Houlding, my project supervisor, for his guidance and invaluable advice throughout this project without whom, my code would likely still not be running. I would also like to thank Paul, Andrew, Rodrigo and Tom at NewsWhip for allowing me to play with the incredible data they gather and giving me the opportunity to gain an insight into this fascinating and evolving field.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>NO.</th>
<th>SECTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INTRODUCTION AND SUMMARY</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.1. The Client: NewsWhip</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.2. Project Background</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.3. Terms of Reference</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.4. Chapter Summaries</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>DESCRIPTION OF WORK DONE</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>2.1. Problem Definition</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>2.2. Analysis of Original Model</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>2.3. Implementation Options</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>2.4. Review of Existing Viral Measurements</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>SYSTEM OVERVIEW</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>3.1. Model Objectives</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>3.2. Underlying Principles</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>3.3. Analysis Methodology</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>3.4. Technical Environment</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>3.5. Critical Analysis of the System</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>3.6. General analysis</td>
<td>21</td>
</tr>
<tr>
<td>4</td>
<td>CONCLUSIONS AND RECOMMENDATIONS</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>4.1. Conclusions</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>4.2. Recommendations</td>
<td>23</td>
</tr>
</tbody>
</table>
## APPENDICES

<table>
<thead>
<tr>
<th>NO.</th>
<th>SECTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.</td>
<td>Original Project Outline</td>
<td>A.1</td>
</tr>
<tr>
<td>B.</td>
<td>Interim Report</td>
<td>B.1</td>
</tr>
<tr>
<td>C.</td>
<td>Data Overview</td>
<td>C.1</td>
</tr>
<tr>
<td>D.</td>
<td>R Code</td>
<td>D.1</td>
</tr>
<tr>
<td>E.</td>
<td>R Packages Used</td>
<td>E.1</td>
</tr>
<tr>
<td>F.</td>
<td>Model Performance</td>
<td>F.1</td>
</tr>
<tr>
<td>G.</td>
<td>The Mathematics of Virality</td>
<td>G.1</td>
</tr>
</tbody>
</table>
1. INTRODUCTION AND SUMMARY

This chapter describes the client, outlines the project background and provides the Terms of Reference while also outlining the subsequent chapters.

1.1. The Client: NewsWhip

NewsWhip tracks the social distribution of the world’s news, creating tens of millions of data points around thousands of stories every day as they measure how news and ideas spread on social networks. They track millions of news articles every month and the social media interactions for them. Meta data such as publisher, category and social platform for each story is also gathered and used to establish the social media impact of these stories. This data is the basis for their “Spike” platform, which delivers a rundown of the most popular news on the web to newsrooms. This platform is being constantly updated and refined and attempts to be the most up to date platform for social media news trends.

1.2. Project Background

Social media analysis is becoming an increasingly relevant field of study in business (Kaplan and Haenlein 2010), marketing (Mangold and Faulds 2009) and media (Kwak, Haewoon et al. 2010). Due to the relative youth of sites such as Facebook and Twitter (the focus of NewsWhip and this project), there is very little academic work exploring the subject. As such there are no standardized measurements of “virality” for social media. Better understanding of the nature of how news spreads on these sites is of great value to those who need or would like to know what the next big thing will be. Being on the cutting edge of what is happening online is of great importance for NewsWhip and their clients. Predictive analytics allow a journalist or news organization to remain ahead of the curve when it comes to social media.

This Project aims to provide an alternate method of finding “trending” articles.

1.3. Terms of Reference

The client is looking to explore predictive methods of measuring article trend and potential trend. This will require;

- Analysis of the patterns of social media activity.
- Design of model to predict "trendingness" for news article.
- Analysis of model to test if implementable.
- Ensure Model is sufficiently flexible to change with social media landscape if necessary.

1.4. Chapter Summaries

Chapter 2: System Overview; this chapter gives an overview of the operations of the model and the technical environment in which it operates. The objectives and evaluation criteria for the model are outlined alongside the underlying mathematical principles applied. Finally, the model is analysis and critiqued.

Chapter 3: Description of Work Done; this chapter describes work done around the project. This includes an analysis of how articles are classified, some existing viral measures and a
more practical look at how a model such as the one described in Section 2 could be implemented in this technical environment.

Chapter 4: Conclusion and Recommendations; here the conclusions of the project are outlined along with a final recommendation for the future of the model designed.
2. DESCRIPTION OF WORK DONE

This section outlines the work done throughout the project.

2.1. Problem Definition

Virality is a term, which, in this context, primarily relates to organic content growth. While news publishers will have their core readership and readers they gain from advertising, a new audience source is social media. People on social media sites are often actively looking for content to engage with and consume. As such, publishers have a lot to gain from these potential customers. This complex mathematical relationship between publishers, readers and their networks is explained in Appendix G.

As outlined in Section 3.4, the more Facebook and Twitter activity a story sees, the more likely it is that it will appear on a user's news or Twitter feed. This leads directly to higher traffic and page views for publishers and as a result, more advertising revenue.

The value to publishers from NewsWhip is that when a new story breaks or trend develops, the client can be made aware of it as soon as possible. There are two sides to this advantage; breaking news and the bandwagon.

If a story is breaking that is quickly becoming a major news item, such as political scandal or natural disaster, then the news agency should be made aware of its existence. This ensures first and foremost that the publisher has not missed this breaking news. It also allows them to see the type of coverage that is getting the most attention so they can, if they desire, tailor their content to a different crowd.

The question remains as to how many stories a publisher is willing to examine in these capacities. This is a major determining factor in deciding classification cutoffs. For many clients a high False Negative rate is acceptable, as they would rather look at more stories than miss an important or interesting one. However, as clients needs vary, the model must be able to adjust to these preferences.

2.2. Analysis of Original Model

NewsWhip utilize two distinct measures of how active a story is. In the first hour, they simply look at the nominal count of interactions with an article. This is presented to users in the form of an activity count over the time period since the first activity.

After the first hour however, they examine velocity of activity or activity per a given time period. Figure 2.1 shows that after one hour (the grey vertical line), small sections of time are considered, indicated by the red line at Time Index 100 to 140, spanning two hours. This would also be presented to customers in the form of an interaction level over a stated time period.
This does not relate directly to the dissimilarity model in this report for one simple reason; the observations must be smoothed. Because the observations in the data are so noisy, examining velocity between any two API checks is redundant and tells us very little about the bigger picture. As such, the velocity must be smoothed over a larger period of time. NewsWhip smooth the data over anything between 30 minutes and 3 hours. Considering the dissimilarity model attempts to operate in small time periods, it simply would not have the necessary amount of information to smooth effectively.

The exact method that is used after the first hour was not available for comparison to the dissimilarity model as NewsWhip preferred to keep that information internal. As such, the nominal count model has been assumed.

2.3. Implementation Options

This analysis was performed by bulk exporting data from NewsWhip’s database and examining the available articles on a separate machine. While this was perfectly acceptable for testing and experimentation, it is insufficient for full integration.

As outlined fully in Section 3.4 all of NewsWhip’s data is held in the cloud in an Amazon Web Services (AWS) Elastic Compute Cloud (EC2). There are two implementation options for how to effectively use this method in this technical environment.

Direct R Implementation: R can be used with Java in a direct way using the rJava package. This allows basic R functionality to run in a Java environment and for R to access objects
and methods that exists in that environment. While this would be a simple way to make the existing code run, it is inefficient.

Recoding: There are languages that offer computation speed improvements over R and are more directly applicable to NewsWhip’s technical environment. Directly coding in java offers the most direct access to the database and also, is more efficient than R.

This can be explained by comparing the strengths and weaknesses of a compiled language (Java) to an interpreted one (R). Compiled languages offer faster performance because they directly interact with the native power available on the machine. Interpreted languages have to be interpreted by some other program before they can be executed, which takes time. Testing and developing the model is almost always a better idea in interpreted languages as they provide results faster in a real world sense for proof of concept. Compiled language is necessary in the long run.

Recoding in Java also solves the RAM issue. R performs all of its calculations in a very RAM intensive way. As datasets grow, the need for RAM increases. A solution to this problem is to interact directly with the database, item by item, to lessen the required amount of data to be held in RAM at any given time. Java allows us the flexibility to do exactly this. There are three datasets used in this analysis which will be referred to as A, B and C. These are briefly outlined in Appendix C. Dataset B, the largest in this example, contains 10,800 articles and occupies almost 450MB before any calculations are done or results stored. A scalable solution would need to be an out-of-memory solution.

2.4. Review of Existing Viral Measurements

There are several models that presently track social media activity as it pertains to sharing content. Here some of them are outlined and examined.

Facebook Edgerank

Edgerank is a tool designed by Facebook that governs what is displayed, and in what order, on a user’s newsfeed (Widman, 2013). The value of an interaction is somewhat subjective. For the purpose of this project, generating new impressions and clickable opportunities for users is what will potentially increase business. As such, what determines the content of a Facebook Newsfeed?

The algorithm is deceptively simple;

\[ \text{Edge value} = \text{Affinity} \times \text{Weight} \times \text{Time Decay} \]

An edge is an action on Facebook. When a user likes or comments or posts a status update or photo, that edge is of a certain level of interest to another user. The Edgerank algorithm attempts to measure the other user’s interest level based on the type of activity and the nature of past interactions.

Affinity refers to the level of interaction a user has had with another entity (user or brand). It is used as a measure of the relationship with the entity the user has. Commenting, liking, sharing or messaging can influence a user’s affinity score.
Weight is a score assigned to possible actions that deems them more or less valuable. It is a metric of the degree of engagement an action requires. A like is simple to give and requires little to no involvement and as such has a low weight. Comments are more highly rated because they require more active engagement with the “edge”. Generally speaking, the weight can be a surrogate for time taken to complete an action.

As an “edge” ages, it loses value. The Time Decay de-values content exponentially as time increases.

Beyond these three basic factors, “Last Actor” and “Story Bumping” impact newsfeed rankings (Edwards 2013). Last actor attempts to capture the present mood of the user. It assigns a greater affinity ranking to the last 50 entities (friends, companies etc.) that the user has interacted with. Because of this, consistent interaction with customers is essential for driving organic growth of news content.

Story Bumping detracts from the impact of time decay. This feature shows the user not only the newest stories but any story that the user may have missed since last time they logged on that have since garnered significant attention. This does not apply to advertised or promoted content, only to organic content. This means that an old story has a secondary opportunity to be featured on a users newsfeed.

**Twitter Trends**

At all times, twitter tracks “trending topics”. These can be names, phrases or hashtags that are highly used in a given location at that time. Unlike Facebook’s Edgerank model, “who” posts on twitter is not relevant when it comes to twitter trends. The volume of tweets containing the topic are counted and time decayed, resulting in a “top 10” list in any given city or country or worldwide. Most trending topics last around 40 minutes.

This model does not refer to any single article or web page and as such is not as applicable to this project as other measurements of social media activity.

**Buzzfeed “Social Lift” charts**

Buzzfeed is a viral focused media publisher which has had great success in the social media sphere due to its simple design, intriguing headlines and content aimed at demographics most prevalent on social networks. It has around 53 million unique users every week. Every story published has a “social lift” graph that displays to the user the amount of traffic from social networking sites (Buzzfeed 2014). The number is calculated by:

\[
Social\ lift = \frac{Viral\ Views}{Seed\ Views} + 1
\]

This model is deceptive however as it deems any clicks from social networking sites as “viral”. It does not consider the case where a publisher will post their own material to Facebook or Twitter and a user clicks through. This does not constitute the grassroots increase in viewership that virality usually indicates.

Buzzfeed are using this as a primary selling point for advertising partnerships.

**Mashable Velocity Graphs**

Mashable accompany every article with a social media “velocity graph”. According to Chris Heald, the Chief Architect of Mashable (Heald 2013) the model is a simple visual
representation of online spread. This also drives their homepage placement. It uses predictive analytics (which are proprietary) to determine the stories that are believed will have most success in the coming hours and give them prime homepage placement. This is only model in mainstream use that claims to be predictive.
3. SYSTEM OVERVIEW

This section outlines the objectives of the model, the underlying principles that the final model is built on, describes the operation of the system and analyses its positive and negative qualities.

3.1. Model Objectives

The model has one primary objective, to predict potentially viral articles using small amounts of social media activity. Virality is a concept that has no formal definition. As such, a custom definition has been created for the purposes of this model, which is outlined in Section 3.3 p. 11. Presently, NewsWhip utilize two methods of determining if an article is deemed “viral” or not which are discussed in Section 2.2. This model will attempt to work better than or in conjunction with the NewsWhip model used in short time periods.

This objective must be achieved in context however. Fast operation is key as NewsWhip’s clients are constantly trying to stay ahead of the curve. Long calculation time loses the value that could be gained from early detection.

The model should present either more, or different articles than the existing model. In accordance with Section 2.1, the model places more importance on a high rate of viral articles found (specificity) instead of model precision. A high False Positive rate is acceptable.

3.2. Underlying Principles

The underlying principle of the final model is based on time-series dissimilarity as a measure of similar behavior patterns.

Time Series

The Facebook and Twitter data is gathered in a time series format. A time series is a sequence of observed data points over time. Most time series would have regular intervals of observation and length; the data gathered by NewsWhip does not, which presents us with an issue of comparison. An example of a time series from an article can be found in Appendix C page 3.

Interpolation

Comparing dissimilar time series can be problematic. For this data set, the checks for individual articles are made at different times and all articles have unique check times. This is due to the fact that there are two distinct sources for the information, the Facebook and Twitter API’s which are discussed in Section 3.4. This means comparisons between Facebook and Twitter activity require interpolated values. Linear Interpolation effectively constructs a new data points between two known observations.

If we had two Facebook observations at times $x_1$ and $x_3$ and we wanted to compare Facebook activity to that at time $x_2$, where we find a Twitter observation, we would have to estimate the Facebook activity at $x_2$. This is done by solving for $y_2$ in the following equation:

$$y_2 = \frac{(x_2 - x_1)(y_3 - y_1)}{(x_3 - x_1)} + y_1$$
Dynamic Time Warping with Euclidean Distance

Dynamic Time Warping (DTW) is a name for a class of algorithms specifically designed for comparing time series (Giorgino 2009). The basis of DTW is that two time series can be manipulated (through stretching and compressing) to make them as similar as possible. The distance between the two series is the sum of the individually aligned elements.

Mathematically, we want to compare a test, \( x = (x_n, ..., x_1) \) and a reference \( y = (y_n, ..., y_1) \) time series. The method assumes that a local dissimilarity function \( f \) is defined between any pair of elements \( x_i \) and \( y_j \) according to:

\[
d_{ij} = d(x_i, y_j) = f(x_i, y_j) \geq 0
\]

A distance matrix, \( D \), of vectors \( x \) and \( y \) is the only input necessary for the function to work. The dissimilarity function used is Euclidean distance.

Euclidean distance is one of the most common measurements of distance between two objects. It follows the general form:

\[
d(p, q) = d(q, p) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2} = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}
\]

This is primarily used for measuring point distance and as such must be used as part of a larger algorithm for time series dissimilarity. Other distance metrics are available to be used, such as Manhattan and Squared Euclidean are available within this algorithm, from the R-package dtw (Giorgino 2009), but Euclidean is the fastest.

The fundamental element of the technique is the warping curve \( \phi(k), k = 1 \ldots T \):

\[
\phi(k) = \left( \phi_x(k), \phi_y(k) \right) \text{ with }
\phi_x(k) \in \{1 \ldots N\},
\phi_y(k) \in \{1 \ldots M\}
\]

The functions \( \phi_x \) and \( \phi_y \) warp and remap the time indices for \( x \) and \( y \) respectively. With \( \phi \) we are able to compute the average accumulated distortion between the newly adjusted \( x \) and \( y \):

\[
d_\phi(x, y) = \sum_{k=1}^{T} d \left( \phi_x(k), \phi_y(k) \right) m_\phi(k) / M_\phi
\]

Where \( m_\phi(k) \) is a weighting coefficient for each pairing in the process and \( M_\phi \) is the corresponding “normalization” constant which is determined by dividing parameters by the overall standard deviation of the entire datasets used in the comparison. This constant makes sure that the summed distortions are comparable along different paths.
To guarantee logical warps, constraints are often imposed on the warping function $\phi$. Commonly, *monotonicity* is used as a constraint to ensure the observations remain in time order:

$$
\phi_x(k + 1) \geq \phi_x(k) \\
\phi_y(k + 1) \geq \phi_y(k)
$$

Ultimately, DTW is a minimization problem. It aims to find the optimal value of $\phi$ such that:

$$
D(x, y) = \min_{\phi} d_\phi(x, y)
$$

DTW algorithms are primarily used in speech and gesture recognition when trying to match an input to various pre-defined sounds or movement patterns. The *dtw* package (Giorgino 2009) contains many implementations of the algorithms and is outlined in Appendix E. Figure 3.2 shows a warped distance example for two time series.

**DTW Example**

![DTW Example](image)

Figure 3.1 Dynamic Time Warp Example

The object created by the *dtw()* function contains much information. We only extract the $\textit{distance}$ object however. This is an un-normalized distance measure. Usually, normalization is necessary for partial comparisons and comparisons of differing lengths. Considering we have standardized our data, this step is not necessary.

It is important to note that due to the warping that has taken place, the distance measures have no real intuitive meaning. They cannot logically be expressed in a form like "activity per time-period". For this reason, the DTW measure can only be applied in a relative way, comparing results instead of selecting from an empirical principle. The boxplot in Figure 3.2 shows the weighted average DTW distance between a test set of articles and a Training set of known viral articles. There is a very clear distinction between the two implying that DTW distance is an effective method of classifying viral or non-viral articles.

![Distance Boxplots Tw](image)

Figure 3.2 Boxplot of weighted Average Dissimilarities for Test Articles
3.3. Analysis Methodology

There are five phases to using the system. These include reading and cleaning the data, manipulating the data, finding the most viral articles, finding article dissimilarity and predicting viral articles. This section outlines the reasoning behind each of these stages and how they are implemented. Examples will all be discussed using a dataset of 10,800 articles that will be referred to as Dataset B.

Phase 1: Data Input, Cleaning and Manipulation

The first task is to read the data into R (R Core Team 2013). The data is presented in Java Script Object notation or JSON format (Crockford 2006). This is a text based, language independent method of transferring data and is explained further in Appendix C. This format can be read using the importJSON function (Couture-Beil 2013) (Appendix D.1). Because the database outputs the information effectively as a single line of text, a new line will need to be added in a text editor to make the text readable. This function reads data in a hierarchical List data type, which is explained in Appendix C, page 3.

The data is now available to be cleaned. Because we are using real world data, it is unreasonable to expect every piece of information to be accurate. Many R functions work in such a way that a single faulty value can completely halt a process. It is more effective to clean the data from the outset, than to attempt to build various bespoke error-handling measures for potential problems when they arise.

There are several recurring problems within the data.

- Data with a single check on the level of Facebook or Twitter activity, providing to little information for a time series.
- Absence of either a Facebook or Twitter check.
- Illogical numbering (unrealistically high or decreasing activity).

These errors are discussed in more detail in Appendix C. While these errors are infrequent in the first data set and almost non-existent in the subsequent two, the removal of these serves primarily as a safeguard even if it may no longer be fully necessary.

There are also fully functional but irrelevant articles. The aim of this model is to find a way of using article time series to predict virality. As such, articles that don’t contribute towards this are disregarded to keep memory usage to a minimum; the reasons for this are discussed in Section 2.3.

Several variables can be removed or simplified. The discovery time of each article is stored across a number of variables as text strings. These can be converted to a much more useful numeric timestamp. Finally, we can disregard any articles that have no social media presence.

Phase 2: Data Manipulation

The data in List format is difficult to analyse, even after cleaning. This is mostly due to the fact that so many important pieces of information are of the String data type and that repeatedly looping through List data can be slow in R. This phase converts these to more usable matrices.
The first step is to create a matrix of Facebook activity and Twitter activity independently. Because the data is gathered through separate APIs (discussed further in section 3.4), the times for various checks are out of synchronisation. Creating a merged matrix and interpolating values can solve this problem. A matrix with six columns can store information pertaining to time of observation, Twitter shares at this time, Facebook likes, comments and shares, and time since article discovery.

The data is collected at inconsistent intervals. This makes time series analysis more difficult. Once the two streams of information have been merged, the missing values for various times must be interpolated. This method is discussed in Section 3.2, page 8 and shown functioning in Appendix D. This has standardized the activity checks within an individual article.

The third and final step is to standardize the checks between articles. Because of the aforementioned inconsistency in activity checks, no two articles will have evenly spaced data points. To solve this problem, time “bins” are created that every article must conform to. These bins represent three minute time periods. The final result of this (after further interpolation) is that every measure of social media activity will be consistently spaced and as such, easily comparable between articles.

**Phase 3: Find Viral Articles**

The next step is to set a bar as to what a “viral” article is by the end of its lifetime. As discussed in Section 2.1, there is no standard definition of what viral actually means in numerical terms. As such, a custom definition will be used. Of the articles with any amount of social media activity, the top 10% of the training set will be deemed to be of viral status. Any article with a level of activity above the minimum of this “Top 10%” group will be deemed viral.

The data is at this point split into test and training sets in a ratio of 1:2 respectively. The various activity measures are sorted and the value of the article 10% from the top is considered the “cutoff”. The cutoff values for Dataset B can be seen in Table 3.1.

<table>
<thead>
<tr>
<th>Activity Type</th>
<th>Minimum Cutoff Activity Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twitter Shares</td>
<td>47</td>
</tr>
<tr>
<td>Facebook Shares</td>
<td>33</td>
</tr>
<tr>
<td>Facebook Likes</td>
<td>55</td>
</tr>
<tr>
<td>Facebook Comments</td>
<td>24</td>
</tr>
</tbody>
</table>

Table A.1 Cutoff Activity Levels

Creating a set of all the articles in the training set with values above results in four vectors of articles that we know to be viral and can compare test articles to. The early activity (up to one hour) will be used to predict if an article is viral by the time it stops being shared on social media.

**Phase 4: Find Article Dissimilarity**

This phase takes the entire test set and measures their dissimilarity, for a certain time period from the first social media activity, against the same time period for the top articles from the train set. This is done according to the Dynamic Time Warping (DTW) model outlined in Section 3.2. In total for Dataset B this means 724,636 calculations will be done. This matrix needs to then be simplified to single usable values for prediction. This requires the creation of average dissimilarity, weighted average dissimilarity and nominal activity vectors. The method by which these are made can be found in Appendix D.
Phase 5: Predicting Viral Articles

There are two main ways that predictions can be output.

1. Each article now has a single value for dissimilarity, a cutoff value can be selected and all articles with a cutoff value lower than that will be presented as viral. The lower the dissimilarity known viral articles, the more likely a new article is to end up viral. This is similar to a single node classification tree.

2. X number of proposed articles can be specified and the X articles with lowest dissimilarity and highest nominal activity will be output by each of the models.

As discussed in Section 3.5 page 18, the nominal model is generally the most accurate, closely followed by the weighted average model. Additional articles that are unique to the weighted average dissimilarity list are output separately. The efficacy of these methods is discussed in Section 3.5 and in greater detail in Appendix F.

Finally a CSV file is output that contains the URLs for the requested articles in two columns, the first containing the results from the nominal model, the second the results from the weighted average model. A diagram of the process can be seen in Figure 3.4.
3.4. Technical Environment

\( R \)

This analysis used R (R Core Team 2013). R is a free and open source software environment for statistical computing and graphics. It is an evolution of the S language. R runs on UNIX, Windows and Mac OS systems.

R is not a compiled programming language in the same way as Java or C, but functions can be defined and it can be altered and implemented through other languages. Because the analysis done for this report was a proof of concept, there was no need for a user facing Graphical User Interface (GUI). Instead, raw information was output and presented graphically to the client in the form of manually constructed reports and updates.

The results are output as a CSV file in addition to being displayed in the R console. These CSV files were read and sometimes used for graphing in Microsoft Office Excel.

\( \text{NewsWhip} \)

The article discovery process relies primarily on RSS (Rich Site Summary) (Lerner 2004) technology and also some web crawling.

RSS is a format for delivery of changing web content, for this reason it is often referred to as “Really Simple Syndication”. It is designed to let a user know about any change on a site and allows users to receive timely updates without having to manually check the site.

NewsWhip effectively run a large scale RSS aggregator for the sites they track. This leaves them with the URLs to be tracked throughout the day and analyzed.

Web Crawling is a page discovery mechanism. The best example of a web crawler is the “Googlebot” (Cutts 2010). This is the method by which Google index web pages to make them available for search in its database. It involves taking a number of starting web pages, going to every link on those pages, and then repeating the process building an index of pages.

The NewsWhip run a Cassandra database on an Amazon EC2 server. Amazon EC2 (Elastic Computer Cloud) is a commercial cloud computing service from Amazon Web Services (AWS). It allows large scale, flexible virtual systems to run on as many systems as are necessary for smooth operation.

NewsWhip use a Cassandra database to store their data, which outputs JSON (JavaScript Object Notation) data types. Cassandra is an open source database management system designed by Apache. Its linear scalability makes it the ideal database type for NewsWhip’s large scale of data gathering.

JSON is often used as a replacement for XML, the Extensible Markup Language (Bray 1998) to transmit data from a server to an application. It is a data type that does not lend itself well to statistical analysis, but is necessary to capture the depth of data in each article tracked.

Tomcat is the container in which Java is run as the primary language for database interaction. Several instances of these can be run depending on the demands being placed on the system.
Twitter and Facebook API

The Twitter and Facebook APIs (application programming interfaces) are tools that allow third-party developers to incorporate some elements of their service to build programs on. The two APIs operate somewhat differently but still provide the same general information. NewsWhip queries the particular URL it is attempting to derive information about and is returned with the raw numbers of how many interactions that URL has had on the network.

Twitter’s API is based on a REST (Respresentational State Transfer) architectural back-end. This ensures that the service works with most syndication formats. Web syndication refers to the simple process by which an application gathers information from one source and sends it to another. RSS is an example of a Web Syndication format and is compatible with Twitter’s API.

The Facebook “Graph” API allows developers to query the number of likes, comments and shares on a URL. It does not however, grant the developer access to private profiles that have made these interactions making deeper analysis very difficult for third parties.

3.5. Critical Analysis of the System

The system has fulfilled the primary objective outlined in Section 2.1 and presented an alternative model of extracting viral articles for NewsWhip’s clients. This section will discuss how effectively the system operates in the context of trade-offs between viral pickup rate, the acceptable number of false negatives, and computation time.

Time

As outlined in Section 3.1, needs vary from client to client. Generally speaking however, NewsWhip’s clients prioritize time of discovery most highly. Three factors determine how early a viral article can be identified. The first is NewsWhip’s discovery of that article; the process by which this is done is discussed in Section 3.4. This element is beyond the control of the model but testing the level of activity at discovery time indicates that late discovery is an issue in less than 2% of articles.

Assuming that the article has a partial time series that the model would classify as viral, the time taken to make that calculation should be as small as possible to ensure early transmission of such information to the client.

The DTW calculation itself and the number of calculations must then be considered as the primary determining factors in time taken. The DTW algorithm, outlined in Section 3.2, runs efficiently, but is complex. The time being considered is a factor that contributes to its run time. Considering we are dealing with such small time periods, small adjustments still have an impact on speed. Figure 3.2 shows how changing the time period being analysed affects run time.
As can be seen the time taken to complete the task increases with more time being considered. Visually, the change looks to be polynomial in nature. While the time taken increases, it is not by a great amount. To find this exact relationship a regression analysis was run on the data. Once this is done we can use the regression formula to predict how long a calculation will take, should the number of articles being considered remain the same.

This change fits the general equation:

\[ y = 0.0002x^2 + 0.0006x + 19.184 \]

The second factor determining how long the model takes to run is the number of articles being compared. The tests on the above example were performed using Dataset B, which contains 10,800 articles. After cleaning and splitting, there are 383 training articles and 1,892 test articles. As such, the above times indicate the time taken to perform 724,636 DTW calculations.

Reducing the number of “train” articles to compare with a test article will linearly reduce the time taken to compute the average dissimilarity because there are fewer calculations to perform. Figure 3.3 shows exactly how this relationship changes. There is a linear relationship between number of calculations being made and the time taken to complete. Which can be expressed in the equation \[ y = 1.7025x + 0.8907 \].

Comparing the maximum amount of possible articles maximizes accuracy and model efficacy. The computation time may be too long however and a trade-off may
have to be made depending on client requirements.

There are a number of potential changes that would reduce calculation time:

- Use a sample of training data articles available.
- Only compare article to others in the same category.
- Use the average curve of all train and do a single calculation.
- Combine concepts and only use the average level of articles in the relevant categories.
- Changing to a faster programming language (Discussed in Section 2.3 page 3)

The effect each of these changes make to accuracy and time are outlined in Appendix F.

**Pickup Rate and False Positives**

There are two ways of presenting the results of this model as outlined in Section 3.3; sufficiently small distance and number of proposed articles.

With both of these methods, we have a high level of control over the sensitivity and specificity of the results. Consider the 30-minute classifications for Dataset A. Figure 3.4 shows the change in recommendation accuracy (True Positive Rate) and viral article pickup rate (the proportion of available viral articles to be found) for various cutoff distances.

What this graph tells us is that a client can have either a very high level of accuracy or a high pickup rate for viral articles, but not both. If an organization has a high tolerance for False Positives it can set the cutoff rate at 26 (it is useful to re-iterate that these warped measures have no real intuitive meaning). In this case it will be presented with 172 articles, 47 of which will ultimately be viral. The model has found 87% of the available 54 viral articles but the ratio of bad articles to good in the final list is 2.66:1.

Conversely, an organization may only want to see articles that are guaranteed to make an impact and may set its cutoff to 22. This client will be presented with a list of only 33 articles, 28 of which will be viral by the end of their lifespan, a recommendation accuracy of 85%.

This example illustrates the fundamental trade-off that exists in the model. Full analysis of the effects of how many articles we choose to be presented with or the cutoff level is done in Appendix F.
Comparison to Existing Model

The fundamentals of the existing model in use by NewsWhip can be found in Section 2.2. They use only the nominal count of activity up to a certain time period to classify articles. Here the dissimilarity model is compared to the existing one.

For purposes of illustration, each model will be asked to propose the same number of articles from its output. The number of articles proposed that ultimately become viral will be calculated and the model’s accuracies will be plotted against each other.

Figure 3.7 Model Comparison Graph

Figure 3.5 plots the number of proposed articles against the accuracy for various versions of the model. The green line is the nominal model that is presently in use. The closer the line is to the number of proposed articles, the more accurate it is. The nominal model is the most accurate considered here. It slightly outperforms the weighted average DTW distance model. It is also clear that both models being considered are maintaining near perfect accuracy up to approximately 50 recommendations. This is due to the fact that 30 minutes after the first activity, some articles have already exceeded the threshold of “virality” and are easy to classify, others are heavily indicating that they will exceed that threshold or are very close to it and both models recognize that and classify accordingly.

Full detailed analysis of model comparisons can be found in Appendix F.
SWOT Analysis

While computation time and model accuracy are important measures of how well a system performs, it is also useful to examine the non-technical elements of the model. Here a Strengths, Weaknesses, Opportunities and Threats (SWOT) model (Pickton and Wright 1998) was used.

Strengths

*Flexibility:* Clients have varying needs and these are all catered for within this method of classification. Those who value time above all else can sacrifice some degree of accuracy to find the stories they want, those who wish to only know the most important stories can filter to a high level of accuracy and conversely, those who want a wider view of social media at any given moment can get it if they are willing to accept false positives.

*Accuracy:* The model is effective at its primary task of finding viral stories. This level of success is detailed and questioned in Appendix F.

*Previously Unknown Articles:* While the system operates at a similar level of efficacy to the existing one, there is not complete overlap within them. The dissimilarity model finds articles that the nominal model would overlook. While these articles can't be recommended with the same level of certainty they do provide a widened view of the social media landscape at the time.

Weaknesses

*Black Box Approach:* There is no first principles proof of why this model works. Experimentally we can demonstrate its efficiency but this may be insufficient for some clients who would like more information about why they are being served the articles they are. This is particularly due to the fact that the warping done in the DTW is lost. The information is being adjusted and changed and that information is not being relayed directly in the final output.

*Unit of Measure:* A further issue with DTW it that is has an unintuitive unit of measure. The final distance measurement does not coerce to an “activity per time” measurement that is easily understood. As such it can be difficult to select cutoff points for article recommendation. Again, experimental knowledge of what True Positive Rate and False Positive Rate these cutoffs give serve as an effective surrogate.

*Speed:* The time taken to compute distance across every available training article is problematic. Many potential solutions to that problem are available however and outlined in Appendix F.

*False Starts:* The tests performed on this model assume the first interaction will trigger virality. This is simply not the case. Many publishers will push their content on social media sites after the time of print, sometimes in the middle of the night or very early in the morning. As such we would see a social media interaction, but no real-life user will interact with it for several hours.

This assumption is easily overcome by making the articles eligible for re-examination after an extended period of no activity. This problem also exists in the nominal model however, meaning that the flaw is balanced out for comparison purposes.
**Opportunities**

**Growth of Training Set:** As a rudimentary learning model, the more data available to it, the more accurate it can become. Because NewsWhip gather massive amounts of information, only a small percentage of it was used in developing this model. There is scope for more specific and refined implementations of this theory like those discussed earlier in this section and in Appendix F. This decreased the accuracy of the model as less data is used from the training set. The problem with a larger dataset is that it would require more computing time, which is already a challenge. There is also the issue of fundamental changes in the social media landscape. What defines a “viral” story now, may not be the same several months from now.

**Language Change:** R is not the most efficient language to use with NewsWhip’s existing system. Should NewsWhip implement the algorithm, it would likely need to be re-coded into a compiled language such as Java that can interact more directly with the database. This is explained in more detail in Section 2.3.

**Deeper Analysis:** It is possible to gather much more publicly available information about top stories. If it is possible (as it appears to be) to find viral stories early in their lifecycle it allows for deeper analysis beyond sharing. Methods such as Sentiment Analysis (Pang and Lee 2008) could be utilized to present customers with not only the level of engagement, but also the nature of that engagement.

**Threats**

Bots and Click Farms are becoming increasingly prevalent on social media (Arthur 2013). Bots and “Botnets” are automated programs that like and share content on a massive scale. Click Farms consist of low paid workers (primarily in Cairo and Dhaka) who are commissioned to replicate real user interactions for customers such as liking pages and posts on Facebook or sharing and re-tweeting content on Twitter.

Both of these methods can vastly impact a publisher’s content. There is no way for us to distinguish a legitimate like from a fraudulent one. This is especially problematic as the bots and farmers like non-commissioned content also to seem more realistic. This presents a problem not only for publishers who want to determine their true readership, but for advertisers who may be paying per action or page view.

**Disregarded Models**

A fully parametric “early indicators” model was attempted. This involved analyzing components of the time-series such as peaks, pits and velocities for short periods of time after the first recorded activity. This model was over-fit and failed to predict virality at anything close to an acceptable level.

While a weighted average dissimilarity model is used in the final model, a minimum dissimilarity model was also tested. The advantage of using averages is that it can mediate "similar noise". With such noisy data over such short time periods it seemed to find the most similarly noisy data instead the most actually similar. This model also failed to be a useful predictive tool.

A Hierarchical Clustering model was created to work in conjunction with the average dissimilarity model. This failed to make meaningful separations in the data and only served to
limit the number of training articles used for test comparison. Categorization is a more effective method of reducing training set size as discussed in Appendix F.

3.6. General analysis

A Secondary aim of this project was to provide a general analysis on the nature of viral content. The rich data gathered by NewsWhip have presented some interesting insights about story lifecycles, publishers and general article behavior.

Basic Behavior

NewsWhip gather thousands of stories a day. From a dataset this large we can make certain statements about the general level of engagement with online content.

- Approximately 45% of articles never make it to social networks
- Facebook has over three times as many article shares as Twitter
- The average lifespan of an article is 20 hours
- An articles takes on average 90 minutes to reach peak activity from its first activity

Of the 10,800 articles in Dataset B, 6,654 have no Facebook activity and 5,737 have no Twitter activity. 5,124 articles (47%) have neither and as such can be disregarded as irrelevant for model construction. Dataset C has 3,469 of 8,000 articles (43%) with no social media activity.

By comparing the first activity of an article to its peak and final activity, we can get a sense of how long an articles life cycle is. On average, a viral article takes 29.3 time periods (each being three minutes) to reach its peak level of activity and has a 12-minute (four time period) spell of zero activity approximately 30 minutes after that. Once an article peaks, its level of activity seems to fall quickly.

Interactions of Comments, Likes and Shares

There are 4 series of values being considered for each article. Likes, shares and comments on Facebook and shares on twitter. While it is possible to combine this into a generic social “score” or “activity rating” it is possible that this will lose valuable insights into the differences between these four activities. By comparing the peak activity rates for each of these across the most popular articles we can see how they influence each other.

Facebook shares are the earliest to peak, 38 and 48 time periods before likes and comments respectively. As such, we can call them the earliest indicator of article activity. For this reason, the Facebook Shares variable was used ahead of the Likes or Comments to attempt to reduce the time taken to accurately predict an article.

Universal Activity

Because the activity has been classified in time bins, it becomes very easy to see how much activity has taken place across every article in those bins. While the Datasets considered here are only snapshots of what is active in the system at a given time it only covers certain
time periods. As such, with the data available it is not possible to fully show the level of activity over long enough time period to gain any meaningful insights. We can however, look at the limited data we have to demonstrate this concept for further examination.

For many publishers, social media sharing is assumed. The most shared publishers can be seen in Figures 3.6 and 3.7. It is worth noting that the data sets considered cover only a short period of time and a small amount of the publishers we could consider. With a complete data stream it could prove beneficial to include publisher expectation in our definition of virality. By comparing article activity not only to other articles in its category, but other articles from its publisher, we may be able to theoretically improve model efficacy. This is shown not to be the case in Appendix F. We can also use the information gathered to see which publishers perform best on each platform.

The lists contain some of the same names, with The Huffington Post being the most active in the time period considered. We can also see that the publishers with the most articles do not necessarily have social media activity that reflects that quantity. The sources with most articles discovered in this time frame are shown in table 3.2.

<table>
<thead>
<tr>
<th>Publisher</th>
<th>Article Count</th>
<th>Twitter Shares</th>
<th>Facebook Shares</th>
<th>Facebook Likes</th>
<th>Facebook Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.wsoctv.com/">www.wsoctv.com/</a></td>
<td>624</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><a href="http://www.wsbradio.com/">www.wsbradio.com/</a></td>
<td>581</td>
<td>0</td>
<td>24</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td><a href="http://www.wokv.com/">www.wokv.com/</a></td>
<td>573</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><a href="http://www.4-traders.com/">www.4-traders.com/</a></td>
<td>209</td>
<td>72</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><a href="http://www.ad-hoc-news.de/">www.ad-hoc-news.de/</a></td>
<td>136</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table A.1 Publishers with most articles detected
4. CONCLUSIONS AND RECOMMENDATIONS

The chapter outlines the conclusions and recommendations about the newly designed model and its value to NewsWhip and its clients. These are based on the extensive testing of the model.

4.1. Conclusions

There presently exists no standard measure of virality and it remains a fluid concept. The model proposed takes both a wide and narrow view of virality that attempts to compare articles to both the entire set of active stories at the time or alternatively only similarly categorized articles. The “Top Ten Percent” definition (Section 2.3) ensures that the stories selected are of genuine significance in the social spectrum.

The weighted average DTW dissimilarity model serves as a versatile and effective tool for predicting virality. In performance terms, it is comparable to the nominal model. Around 50 minutes after first activity, it begins to outperform the existing model, but this is also around the time when the nominal model ceases to be used and prediction is less useful than measurement. As such there exists only a small window of opportunity in which this model is superior (Appendix F, page 5), it does however fulfill its aim of providing a credible alternative to nominal measurement.

The models secondary advantage is that it identifies potentially viral articles that the nominal model missed. While this has its uses, the accuracy level of these unique predictions is very low, sometimes as low as 10%.

The calculation time and processing power demands are also a major factor in analyzing the model. At present, the most accurate version of the model operates at too slow a pace, although simplification of the method can alleviate this somewhat. Appendix F outlines some of these solutions and their consequences.

Several valuable and interesting insights have been derived from the data available about the nature of trends and social media activity. These do not impact on the model in any significant way but their complete findings are outlined in Appendix G.

4.2. Recommendations

All factors considered, it is not reasonable to recommend this model be implemented into NewsWhip’s operations as it stands. The short-term model in use is superior to the proposed one both in accuracy and in speed in almost every test performed in Appendix F. As such should still be used as the primary method of article classification.

There is scope for a limited implementation of the new model as a supplementary method of finding articles as shown in Appendix F, page 3. Should the computing power be available, more articles would be identified and presented to clients earlier that the original model did not find. In this sense, the model could be said to be a success as it can deliver additional content at an equal accuracy level to NewsWhip’s clients. The question of computation time makes that a very difficult implementation challenge.
A. **Original Project Outline**

This is the original outline for potential work to be done on behalf of NewsWhip.

**Client:** NewsWhip  
**Project:** Refining algorithm for measuring social media activity  
**Location:** Dogpatch Labs, Barrow St, Dublin 4  
**Client Contact:** Paul Quigley – paul.quigley@newswhip.com

**Client Background**

NewsWhip tracks the social distribution of the world’s news, creating tens of millions of data points around hundreds of thousands of stories every day as they measure how news and ideas spread on social networks.

**Project Background**

NewsWhip track million of news articles every month and the social media interactions for them. Meta data such as publisher, category and social platform for each story is also gathered and used to establish the social media impact of these stories. This data is the basis for their “Spike” platform, which delivers a rundown of the most popular news on the web to newsrooms. This platform is being constantly updated and refined.

**Client Requirement**

- Developing a method to define a threshold for “trendingness” for different categories of content.  
- Design a measure of expected activity based on various meta data.  
- Pulling and cleaning down data sets for publication showing the most successful social news sources, journalists, and topics in customer facing reports.  
- Perform quantitative research on the social lifecycle of a news story to help us refine our algorithms, and for possible publication.

**What is involved for the student?**

This project will give the student an opportunity to work hands on with massive amounts of real world data. Refining the story tracking algorithm to include predictive elements, while still maintaining company credibility will be a unique challenge.

Liaising with the lead software engineer and CEO to ensure that any changes made are possible in terms of available resources and provide sufficient return to clients for the resources used.

It is also imperative however that the work being done is sufficiently flexible so that it can keep up with changing customer needs and a dynamic work environment.
B. Interim Report

Project: Trend Prediction Model
Client: Newswhip
Student: Jack Toner
Supervisor: Brett Houlding

Review of Background

NewsWhip tracks the social distribution of the world’s news, creating tens of millions of data points around thousands of stories every day as they measure how news and ideas spread on social networks. They track millions of news articles every month and the social media interactions for them. Meta data such as publisher, category and social platform for each story is also gathered and used to establish the social media impact of these stories. This data is the basis for their “Spike” platform, which delivers a rundown of the most popular news on the web to newsrooms. This platform is being constantly updated and refined and trying to be the most up to date platform for social media news trends.

At present, there is very little academic work done on trend analysis on social media and even very little of it is done with a news focus. This is likely attributable to the relatively young age of sites like Facebook and Twitter (the focus of NewsWhip’s operations and this project). The work that has been done has been in relation to trending topics and sentiment analysis. While both are interesting fields and somewhat relevant to this project, neither deal with the rate at which users are driven to spread external material on social media sites.

Terms of Reference

The client is looking to explore predictive methods of measure article trend and potential trend. This will require;

- Analysis of the patterns of social media activity.
- Design of model to predict "trendingness" for news articles.
- Analysis of model to test if implementable.
- Ensure model is sufficiently flexible to change with social media landscape if necessary.

Work Carried Out to Date

The first task was to clean and re-organize the data. Newswhip store their data in JSON format, which is difficult to analyze numerically because it is read by R as a “List” data type. While this is suitable for storing time-series information it is very difficult to analyze. Initially, a test data set of just over 600 news articles was obtained in list format.

This data also needed to be cleaned. There were several errors in the data such as non-logical values being gathered and missing values. Isolating and removing these was difficult due to the format of the data. It was done by examination. By plotting the time series charts of the articles, the errors became more obvious.

To get a sense of the general nature of news virality in social media I did some preliminary analysis. I found that most news stories do not make it to social media sites, that trend rate increases exponentially and that it decreases linearly. These observations will be essential in the design of the final model.
Planned Work

Over Christmas Break

- Begin testing initial models and decide on method of comparison.
- Refine algorithm for accuracy and efficiency

Hilary Term – Up to Week 9

- Continue to refine algorithm and test accuracy
- Get as much data as possible to test scalability
- Present First Draft solutions to client for review and criticism

Hilary Term – Week 10-12

- Finalise algorithm and present to Newswhip for potential implementation
- Complete First Draft of project report for review by Supervisor
- Complete Project Report
## Data Overview

Below is a description of each of the variables in the data extracted from NewsWhip's database. Not all of these are relevant and some are not present* in all three datasets received.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>chr</td>
<td>Unique character list for identifying articles</td>
</tr>
<tr>
<td>imgHRef</td>
<td>chr</td>
<td>URL linking to main image from article site</td>
</tr>
<tr>
<td>imgWidth</td>
<td>num</td>
<td>Width of the main image</td>
</tr>
<tr>
<td>imgHeight</td>
<td>num</td>
<td>Height of the main image</td>
</tr>
<tr>
<td>headline</td>
<td>chr</td>
<td>The text taken from HTML tags <code>&lt;og:title&gt;</code> or <code>&lt;head&gt;</code></td>
</tr>
<tr>
<td>summary</td>
<td>chr</td>
<td>Text taken from HTML</td>
</tr>
<tr>
<td>twCreator</td>
<td>chr</td>
<td>The First found Twitter user to share the URL</td>
</tr>
<tr>
<td>sourceIds</td>
<td>num</td>
<td>Empty Variable</td>
</tr>
<tr>
<td>scores</td>
<td>List</td>
<td>List of numbers, the string name of each of these is a timestamp.</td>
</tr>
<tr>
<td>maxScore</td>
<td>num</td>
<td>The peak value in the scores List</td>
</tr>
<tr>
<td>fbLikes</td>
<td>List</td>
<td>A list of all of the checks on facebook &quot;likes&quot; for the article.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The name of each list element is a timestamp in String format. The number of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>likes are the num elements within the list.</td>
</tr>
<tr>
<td>fbShares</td>
<td>List</td>
<td>As above with Facebook Shares</td>
</tr>
<tr>
<td>fbComments</td>
<td>List</td>
<td>As above with Facebook Comments</td>
</tr>
<tr>
<td>twShares</td>
<td>List</td>
<td>As above with twShares</td>
</tr>
<tr>
<td>liCount *</td>
<td>List</td>
<td>Empty Variable</td>
</tr>
<tr>
<td>piCount *</td>
<td>List</td>
<td>Empty Variable</td>
</tr>
<tr>
<td>categories *</td>
<td>chr</td>
<td>Topics under which the articles could be categorised</td>
</tr>
<tr>
<td>authors *</td>
<td>List</td>
<td>The writers name, if available</td>
</tr>
<tr>
<td>ignore *</td>
<td>logi</td>
<td>Boolean value indicating if the article is still being considered. TRUE in</td>
</tr>
<tr>
<td>fbDate *</td>
<td>num</td>
<td>Unused time stamp from Facebook</td>
</tr>
<tr>
<td>fbId *</td>
<td>num</td>
<td>Facebook unique ID for this URL to be used in its API</td>
</tr>
<tr>
<td>href</td>
<td>chr</td>
<td>The URL for the article</td>
</tr>
<tr>
<td>discoveryTime</td>
<td>List</td>
<td>A List of 24 num elements that mark the discovery time of the article</td>
</tr>
<tr>
<td>sourceUrl</td>
<td>chr</td>
<td>Same as href</td>
</tr>
<tr>
<td>mostRecentFbComments</td>
<td>num</td>
<td>The final observed number of Facebook comments for this article</td>
</tr>
<tr>
<td>mostRecentFbLikes</td>
<td>num</td>
<td>The final observed number of Facebook likes for this article</td>
</tr>
<tr>
<td>mostRecentFbShares</td>
<td>num</td>
<td>The final observed number of Facebook shares for this article</td>
</tr>
<tr>
<td>mostRecentTwShares</td>
<td>num</td>
<td>The final observed number of Tweets for this article</td>
</tr>
<tr>
<td>mostRecentNewsWhipScore</td>
<td>num</td>
<td>The final calculated &quot;social score&quot; assigned by NewsWhip</td>
</tr>
</tbody>
</table>

* These variables are not present in Dataset A.

---

Table C.1 Description of Variables Exported by NewsWhip
Table C.1 shows the data as NewsWhip exported it. Several variables were added for analysis purposes. These are described below.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>discTime</td>
<td>POSIXct</td>
<td>A single value timestamp replacing the &quot;discoverytime&quot; variable list in the original data</td>
</tr>
<tr>
<td>earliestFbCheck</td>
<td>POSIXct</td>
<td>The time of the First check made to Facebook</td>
</tr>
<tr>
<td>numFbChecks</td>
<td>num</td>
<td>The total number of checks made to Facebook</td>
</tr>
<tr>
<td>lastCheckFb</td>
<td>POSIXct</td>
<td>The time of the final check made to Facebook</td>
</tr>
<tr>
<td>earliestCheckTw</td>
<td>POSIXct</td>
<td>The time of the First check made to Twitter</td>
</tr>
<tr>
<td>numTwChecks</td>
<td>num</td>
<td>The total number of checks made to Twitter</td>
</tr>
<tr>
<td>lastCheckTw</td>
<td>POSIXct</td>
<td>The time of the final check made to Twitter</td>
</tr>
<tr>
<td>checkDurationFb</td>
<td>num</td>
<td>The time, in seconds, between the first and the last Facebook check</td>
</tr>
<tr>
<td>discLapse</td>
<td>num</td>
<td>The time between discovery of the article and the first check to either Facebook or Twitter</td>
</tr>
<tr>
<td>mergedMat</td>
<td>matrix</td>
<td>A matrix of nums that hold the time and activity for both Facebook and Twitter. This matrix has been standardised to three minute intervals</td>
</tr>
<tr>
<td>fbLikesFirstActivity</td>
<td>num</td>
<td>The mergedMat index of the first like on Facebook</td>
</tr>
<tr>
<td>fbCommsFirstActivity</td>
<td>num</td>
<td>The mergedMat index of the first comment on Facebook</td>
</tr>
<tr>
<td>twFirstActivity</td>
<td>num</td>
<td>The mergedMat index of the first share on Twitter</td>
</tr>
<tr>
<td>fbSharesFirstActivity</td>
<td>num</td>
<td>The mergedMat index of the first share on Facebook</td>
</tr>
<tr>
<td>publisher</td>
<td>chr</td>
<td>The main URL of the publisher taken as a substring from sourceUrl</td>
</tr>
</tbody>
</table>

Table C.2 New Variables Added for Analysis

Structure of The Time Series

The time series are stored and transmitted in an inconvenient way. Each of the four activity types are saved as a hierarchical list of numbers. The timestamp for activity checks are the headings of these lists. Because numbers in string format cannot be machine read, they must be converted to a more usable format. An example of this can be seen in Figure C.1.

For this, the POSIX format was selected. POSIX, or Unix, time is a method of expressing instances in time measured in seconds from a defined origin point, the 1st January 1970.

When these times have been converted to numeric expressions they can be saved in a matrix format and worked with more easily. The code by which this is done is available in Appendix D page 4.
Data Types

The data is read from Java Script Object Notation (JSON) files. Generally speaking it is syntax for passing data in name/value pairs. JSON has several advantages that make it useful for this kind of data. The values are named, which means they are intuitive and human-readable. This allows for an easier understanding of the data. This convention also lends itself well to R for string based referencing. JSON, being hierarchical, can hold values within values, this is utilized in holding timestamps as names with related numbers as activity in the

The data is then held in R as a **hierarchical List**. Lists are an effective way of storing complex and detailed content in R. Lists were necessary for these particular datasets because they contained not only article meta-data such as URL and topic of interest, but also, multiple time-series objects, of varying lengths. Lists do have their problems however. Accessing all of the information in a list sequentially cannot be done with the same ease as it can in a matrix or a vector. Loops must be used. This is far from a deal-breaker when it comes to using lists, but R is slow at looping. To illustrate, consider the below code:

```r
A = matrix(as.numeric(1:100000))
system.time({
  Sum = 0
  for (i in seq_along(A)) {
    Sum = Sum + A[[i]]
  }
  Sum
})
```

This takes approximately 0.34 seconds to run. Comparatively, `sum(A)` runs in an immeasurably small amount of time. List loops suffer from this same fate.

Datasets

There are three datasets used in these calculations

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Articles</th>
<th>Size on File</th>
<th>Size as List</th>
<th>Upload Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>6300</td>
<td>53.4 Mb</td>
<td>235 Mb</td>
<td>4.864 s</td>
</tr>
<tr>
<td>B</td>
<td>10800</td>
<td>85.1 Mb</td>
<td>417.8 Mb</td>
<td>8.497 s</td>
</tr>
<tr>
<td>C</td>
<td>8000</td>
<td>68.7 Mb</td>
<td>332.9 Mb</td>
<td>6.637 s</td>
</tr>
</tbody>
</table>

Errors

In Dataset A there were a number of recurring errors that required handling. For caution’s sake, these error-handling measures were kept even though these problems could not be identified in sets B or C.

**Negative or illogical numbering:** There were several cases of high levels of existing activity upon discovery of new articles. These values would often disappear and the activity level would set itself to zero after several observations. One possible explanation for this is that bots had artificially embellished the activity on the story upon its publication and these fraudulent Likes and Shares were removed by Facebook’s security.
Single or few checks: There were some cases of articles being queried a small number of times. With too few observations to construct a time-series, these articles were deemed invalid for the purposes of model development. While these articles were not erroneous, they were irrelevant and removed from the set.

Absence of Facebook/Twitter checks: There were some articles which had NA values for Twitter or Facebook activity. This would occur if the article was not successfully queried on the respective APIs. Because attempting to run operations on a number and an NA resulted in crashes, and because this problem was very rare within the dataset (less than 15 instances), these articles were removed.
D. R Code

This appendix contains the original R code used. They are broken into the various tasks each code section had to perform. Included also are sections of code that are used for general analysis even if they are not used directly in the final prediction.

Import data from JSON files

```
# 01 - Data import
install.packages("rjson")
library("rjson")

json_file <- "~/FYP - Final Draft/Data/articles-2-15pm-to-16pm.json"
# "articles-20130601-20131007-50-day new line.json" - Dataset A
# "articles-2-15pm-to-16pm.json" - Dataset B
# "articles-2-19pm-to-20pm.json" - Dataset C

importJSON <- function(filePath){  # Simplified JSON import function
dataSet <- fromJSON(paste(readLines(json_file), collapse=""))
  return(dataSet)
}

system.time(
data <- importJSON(json_file)  # import dataset
)

sort( sapply(ls(),function(x) {object.size(get(x))}))
```

Remove articles with errors outlined in Appendix C

```
# 02 - Data cleaning

# Phase 1 - Articles missing FB or TW checks

#FB

fbMissingChecks <- 0
for(i in 1:length(data)) {
  if (length(data[[i]]$fbLikes) == 0){  # no Fb check made
    data[[i]] <- NULL
    fbMissingChecks <- fbMissingChecks+1
  }
  i<-i+1
}
cat("FB missing Checks = ", fbMissingChecks)

#TW

twMissingChecks <- 0
for(i in 1:length(data)) {
  if (length(data[[i]]$twShares) == 0){  # no Fb check made
    data[[i]] <- NULL
    twMissingChecks <- twMissingChecks+1
  }
  i<-i+1
}
cat("TW missing Checks = ", twMissingChecks)
```
D.2

###
# remove 0s and few checks - need to make time series, these articles dont contribute
###
fewChecks <- 0
article = 1
while (article <= length(data)){
  if( length(data[[article]]$fbLikes)) < 5 || length(data[[article]]$twShares) < 5) { # need 2 non NA values to interpolate
    data[[article]] < - NULL
    article = article - 1 # to avoid skipping over articles
    fewChecks <- fewChecks + 1
  }
  article = article + 1
}
cat("Few checks = ", fewChecks)
###

Remove redundant variables for memory management purposes

# 03 - Variable Removal
# This section removes redundant or unused variables to make the dataset physically smaller

removeRedundantVariables <- function(dataSet){
  lengthData <- length(dataSet)

  for(i in 1:lengthData){
    dataSet[[i]]$href <- NULL # replica of sourceUrl
    dataSet[[i]]$mostRecentNewsWhipScore <- NULL # Always blank
    dataSet[[i]]$imgWidth <- NULL # irrelevant
    dataSet[[i]]$imgHeight <- NULL # irrelevant
    dataSet[[i]]$scores <- NULL # Always Blank
    dataSet[[i]]$sourceIds <- NULL # not sure what this should mean?
    dataSet[[i]]$maxScore <- NULL # Always 0
  }
  return(dataSet)
}
# View(sort( sapply(ls(),function(x){object.size(get(x))})))
# object.size(data)
data <- removeRedundantVariables(data) # execute above function

Translate discovery time to numeric format

# 04 - Discovery and Latest Time
# This loop creates a discTime and earliest and latest tw and fb check timestamps in POSIX

i = 1
while (i <= length(data)){
  discString <- ""
  discString <- paste0(discString, toString(data[[i]]$discoveryTime$year))
  discString <- paste0(discString, ",")
  discString <- paste0(discString, toString(data[[i]]$discoveryTime$month))
}
while(fbNoSocial)
  dataLength <- length(data)
  fbNoSocial <- 0
  while(i < dataLength){
    if (data[i]$mostRecentFbComments == 0 &
        data[i]$mostRecentFbShares == 0 &
        data[i]$mostRecentFbLikes == 0)
      # Indicates no FB activity
        i <- i+1
    else {
      timex <- names(data[[i]]$fbComments)
      checkTime <- as.POSIXct(strptime(timex, "%Y-%m-%d %H:%M:%S"), tz="GMT")
      data[[i]]$earliestCheckFb <- checkTime
      numOfChecks <- length(data[[i]]$fbComments) # index of last check
      data[[i]]$numfbChecks <- numOfChecks
      timex <- names(data[[i]]$fbComments[numOfChecks])
      checkTime <- as.POSIXct(strptime(timex, "%Y-%m-%d %H:%M:%S"), tz="GMT")
      data[[i]]$lastCheckFb <- checkTime
      timex <- names(data[[i]]$twShares)
      checkTime <- as.POSIXct(strptime(timex, "%Y-%m-%d %H:%M:%S"), tz="GMT")
      data[[i]]$earliestCheckTw <- checkTime
      numOfChecks <- length(data[[i]]$twShares) # index of last check
      data[[i]]$numTwChecks <- numOfChecks
      timex <- names(data[[i]]$twShares[numOfChecks])
      checkTime <- as.POSIXct(strptime(timex, "%Y-%m-%d %H:%M:%S"), tz="GMT")
      data[[i]]$lastCheckTw <- checkTime
      data[[i]]$checkDurationFb <- as.numeric(data[[i]]$lastCheckFb) -
                                 as.numeric(data[[i]]$earliestCheckFb)
      data[[i]]$discLapse <- as.numeric(data[[i]]$earliestCheckFb) -
                            as.numeric(data[[i]]$discTime)
    } # end if
  } # end while
  i <- i+1
  fbNoSocial <- fbNoSocial + 1
} # end while

Remove non-social articles

# 05 - Removal of articles with no social media activity
# Remove Fb and Twitter seperately to give indication of activity levels
i <- 1
dataLength <- length(data)
while(i < dataLength){
  if (data[i]$mostRecentFbComments == 0 &
      data[i]$mostRecentFbShares == 0 &
      data[i]$mostRecentFbLikes == 0)
    # Indicates no FB activity
      i <- i+1
} # end while
Create interpolated matrix of all activity

```r
# TW non-social removal
###
i <- 1
dataLength <- length(data)
twNoSocial <- 0
while(i < dataLength){
  if (data[[i]]$mostRecentTwShares == 0 ){ # Indicates no FB activity
    i <- i+1
    twNoSocial <- twNoSocial + 1
  } else { i <- i+1 }
}
cat("Fb no social = ", fbNoSocial, ",TW no social = ", twNoSocial)
# No Social activity at all
i <- 1
dataLength <- length(data)
noSocial <- 0
while(i < dataLength){
  if (data[[i]]$mostRecentFbComments == 0 &&
      data[[i]]$mostRecentFbShares == 0 &&
      data[[i]]$mostRecentFbLikes == 0 &&
      data[[i]]$mostRecentTwShares == 0)
    # Indicates no activity
    data[[i]]=NULL
    dataLength <- length(data)
    noSocial <- noSocial + 1
  } else { i <- i+1 }
}
cat("No social = ", noSocial)
```

```
# 06 - Create Merged Matrix
# Merge TW and FB observations
# Merged Matrix
library(zoo) # for na.approx
beginRun <- Sys.time()
options(scipen=999)
article <- 1 # this runs from 1 - data.length
while (article <= length(data)){
  if( 
      (length(data[[article]]$fbLikes)) < 5 || 
      length(data[[article]]$twShares) < 5)
    # need 2 non NA values to interpolate
    data[[article]]$mergedMat <- NULL
    data[[article]]$likeMat <- NULL
  else {
    data[[article]]$mergedMat <- NULL
    data[[article]]$likeMat <- NULL
  }
```
# Facebook part

def checks <- length(data[[article]]$fbLikes)-1 # exclude last observation
def fbMat <- matrix(ncol = 6, nrow = checks)
colnames(fbMat) <- c("time", "twShares", "fbComms", "fbShares", "fbLikes", "timeOrig")
x <- 1
while (x <= (checks)){
timex <- names(data[[article]]$fbLikes[x])
fbTime <- as.POSIXct(strptime(timex, "%Y-%m-%dT%H:%M:%S"), tz="GMT")
fbMat[x,"time"] <- fbTime
fbMat[x,"twShares"] <- NA
fbMat[x,"fbComms"] <- data[[article]]$fbComments[[x]]
fbMat[x,"fbShares"] <- data[[article]]$fbShares[[x]]
fbMat[x,"fbLikes"] <- data[[article]]$fbLikes[[x]]
fbMat[x,"timeOrig"] <- NA
x <- x+1 }

# Twitter part

def checks <- length(data[[article]]$twShares)-1 # exclude last observation
def twMat <- matrix(ncol = 6, nrow = checks)
colnames(twMat) <- c("time", "twShares", "fbComms", "fbShares", "fbLikes", "timeOrig")
x <- 1
while (x <= (checks)){
twObs <- as.POSIXct(strptime(names(data[[article]]$twShares[x]), "%Y-%m-%dT%H:%M:%S"), tz="GMT")
twMat[x,"time"] <- twObs
twMat[x,"twShares"] <- data[[article]]$twShares[[x]]
twMat[x,"fbComms"] <- NA
twMat[x,"fbShares"] <- NA
twMat[x,"fbLikes"] <- NA
twMat[x,"timeOrig"] <- NA
x <- x+1 }

##### Merge 2 matrixes

fullMat <- rbind(twMat, fbMat)
fullMat <- fullMat[order(fullMat[,"time"]),]

firstTime <- fullMat[1,"time"]
lastTime <- fullMat[nrow(fullMat),"time"]
numTicks <- (lastTime-firstTime)/180

firstTick <- firstTime + (180-firstTime%%180)
lastTick <- lastTime - (180+lastTime%%180)
tickVector <- seq(from=firstTick, to = lastTick, by = 180)
timeMat <- matrix(ncol = 6, nrow = length(tickVector))
colnames(timeMat) <- c("time", "twShares", "fbComms", "fbShares", "fbLikes", "timeOrig")
timeMat[,"time"] <- tickVector

fullMat <- rbind(fullMat, timeMat)
fullMat <- fullMat[order(fullMat[,"time"]),]
for (obs in 1:nrow(fullMat)){
  fullMat[obs, "timeOrig"] <- fullMat[obs,"time"] - fullMat[1, "time"]
}

### interpolate
fullMat <- na.approx(fullMat[,1:6]) # might be a problem with NAs at
beginning and end

# Remove values not divisible by 180
selectionx <- which (fullMat,"time"%%180 == 0)
fullMat <- fullMat[selectionx,]
for (obs in 1:nrow(fullMat)){
  fullMat[obs, "timeOrig"] <- fullMat[obs,"time"] - fullMat[1, "time"]
}

data[[article]]$mergedMat <- fullMat

# Now, make 4 velocity vectors, each smoothed over 10 observations (30
mins)
}

Split data into test and train sets and find values of cutoff points.

# 7 - split and find cutoff
# split the Data into test and train sets
# also find the virality cutoff

library("dtw")
library("functional") # for removing NAs from dissimilarity matrix

# split top articles into train and test data
testData <- sample.int(length(data), length(data)/3)
trainData <- seq(from=1, to= length(data))[-testData]

# find minimum activity virality cutoff for Twitter
x <- NULL
for (i in trainData){
  x <- append(x, as.numeric(data[[i]]$mostRecentTwShares)) # get vector of
  all final tweet counts
}
sortedx <- sort(x) # put in ascending order
cutoff90tw <- sortedx[round(0.9*length(x))] # find value at 90% # START
WITH THIS ONE!?!???????
cutoff80tw <- sortedx[round(0.8*length(x))] # find value at 80%

## find minimum activity virality cutoff for Facebook Shares
x <- NULL
for (i in trainData){
  x <- append(x, as.numeric(data[[i]]$mostRecentFbShares)) # get vector of all final tweet counts
}
sortedx <- sort(x) # put in ascending order
cutoff90fbShares <- sortedx[round(0.9*length(x))] # find value at 90% # START WITH THIS ONE!?!?!!!!!
cutoff80fbShares <- sortedx[round(0.8*length(x))] # find value at 80%

## find minimum activity virality cutoff for Facebook Likes
x <- NULL
for (i in trainData){
  x <- append(x, as.numeric(data[[i]]$mostRecentFbLikes)) # get vector of all final tweet counts
}
sortedx <- sort(x) # put in ascending order
cutoff90fbLikes <- sortedx[round(0.9*length(x))] # find value at 90% # START WITH THIS ONE!?!?!!!!!
cutoff80fbLikes <- sortedx[round(0.8*length(x))] # find value at 80%

## find minimum activity virality cutoff for Facebook Comms
x <- NULL
for (i in trainData){
  x <- append(x, as.numeric(data[[i]]$mostRecentFbComments)) # get vector of all final tweet counts
}
sortedx <- sort(x) # put in ascending order
cutoff90fbComms <- sortedx[round(0.9*length(x))] # find value at 90% # START WITH THIS ONE!?!?!!!!!
cutoff80fbComms <- sortedx[round(0.8*length(x))] # find value at 80%

Find articles that exceed the virality threshold

# 08 - Find top Articles

# find articles above cutoff rate (top articles) Twitter Shares
topArticlesTrain90tw <- NULL
for (i in trainData){
  if (data[[i]]$mostRecentTwShare >= cutoff90tw){
    topArticlesTrain90tw <- c(topArticlesTrain90tw, i) # top article within the training set
  }
}

# Top articles by FB Shares
###
topArticlesTrain90fbShares <- NULL
for (i in trainData){
  if (data[[i]]$mostRecentFbShares >= cutoff90fbShares){
    topArticlesTrain90fbShares <- c(topArticlesTrain90fbShares, i) # top article within the training set
  }
}###

# Top articles by FB Likes
###
topArticlesTrain90fbLikes <- NULL
for (i in trainData){

if (data[[i]]$mostRecentFbLikes >= cutoff90fbLikes)
    topArticlesTrain90fbLikes <- c(topArticlesTrain90fbLikes, i) # top article within the training set
}

# Top articles by FB Comms
###
topArticlesTrain90fbComms <- NULL
for (i in trainData)
    if (data[[i]]$mostRecentFbComments >= cutoff90fbComms)
        topArticlesTrain90fbComms <- c(topArticlesTrain90fbComms, i) # top article within the training set

Find first activity for each article

# 09 - First activity Finder
# create a variable in each article for the 4 activities
# num index of first activity
# tw first activity
###
for (i in 1:length(data))
    firstNonNa <- min(which(!is.na(data[[i]]$mergedMat[,]"twShares")))
    while (data[[i]]$mergedMat[firstNonNa, "twShares"] < 1
        && firstNonNa < nrow(data[[i]]$mergedMat)-1
        && is.na(data[[i]]$mergedMat[firstNonNa,"twShares"])==FALSE)
        firstNonNa <- firstNonNa + 1
    data[[i]]$twFirstActivity <- firstNonNa
    cat("- ", i)
# endof twitter loop
###
# fbShares first activity
###
for (i in 1:length(data))
    firstNonNa <- min(which(!is.na(data[[i]]$mergedMat[,]"fbShares")))
    while (data[[i]]$mergedMat[firstNonNa, "fbShares"] < 1
        && firstNonNa < nrow(data[[i]]$mergedMat)-1
        && is.na(data[[i]]$mergedMat[firstNonNa,"fbShares"])==FALSE)
        firstNonNa <- firstNonNa + 1
    data[[i]]$fbSharesFirstActivity <- firstNonNa
    cat("- ", i)
# endof shares loop
###
# fbLikes first activity
###
for (i in 1:length(data))
    firstNonNa <- min(which(!is.na(data[[i]]$mergedMat[,]"fbLikes")))
    while (data[[i]]$mergedMat[firstNonNa, "fbLikes"] < 1
        && firstNonNa < nrow(data[[i]]$mergedMat)-1
        && is.na(data[[i]]$mergedMat[firstNonNa,"fbLikes"])==FALSE)
        firstNonNa <- firstNonNa + 1
    data[[i]]$fbLikesFirstActivity <- firstNonNa
    cat("- ", i)
```r
&& is.na(data[[i]]$mergedMat[firstNonNa,"fbLikes"])==FALSE){
  firstNonNa <- firstNonNa + 1
}
data[[i]]$fbLikesFirstActivity <- firstNonNa
cat("- ", i)
} # endof shares loop
###
# fbShares first activity
###
for (i in 1:length(data)){
  firstNonNa <- min(which(!is.na(data[[i]]$mergedMat[,]"fbComms")))
  while (data[[i]]$mergedMat[firstNonNa,"fbComms"] < 1 &&
    firstNonNa < nrow(data[[i]]$mergedMat)-1 &&
    is.na(data[[i]]$mergedMat[firstNonNa,"fbComms"])==FALSE){
    firstNonNa <- firstNonNa + 1
  }
data[[i]]$fbCommsFirstActivity <- firstNonNa
cat("- ", i)
} # endof shares loop

Classify viral articles twitter activity in the test set
# 10a - Classification for Twitter
timeDiff <- 10
positionTime = 1
cat("\n\nSAMPLE SIZE ",sampleSize,"\n")
randomTestArticles <- testData # Using every articel in the test set
classificationMatrixTw <- matrix(nrow = length(randomTestArticles), ncol = sampleSize)
# Create Dissimilarity Matrix of randomTestArticles and topArticlesTrain90tw
####
startTime <- as.numeric(Sys.time())
x = 1
y = 1
for (testArticle in randomTestArticles){
  cat("- ",x)
y = 1
  for (trainArticle in topArticlesTrain90tw){
    if(data[[testArticle]]$twFirstActivity+timeDiff <
      nrow(data[[testArticle]]$mergedMat) && # there are timeDiff checks in the future
data[[trainArticle]]$twFirstActivity+timeDiff <
      nrow(data[[trainArticle]]$mergedMat) &&
    is.na(data[[testArticle]]$mergedMat[data[[testArticle]]$twFirstActivity+timeDiff, "twShares"]) == FALSE
    is.na(data[[trainArticle]]$mergedMat[data[[trainArticle]]$twFirstActivity+timeDiff, "twShares"])) ==FALSE)
    { distance <- dtw(
      data[[testArticle]]$mergedMat[data[[testArticle]]$twFirstActivity: (data[[testArticle]]$twFirstActivity+timeDiff),"twShares"],
```
\[ y = \text{data[mergedMat[trainArticle][twFirstActivity + timeDiff], "twShares"],} \]
\[
\text{dist.method = "Euclidean", distance.only = TRUE, keep.internals = FALSE}
\]
\[
\text{classificationMatrixTw}[x, y] \leftarrow \text{distance$distance}
\]
\[
\text{else} \{ \# \text{if there is not 1 hour to compare}
\text{classificationMatrixTw}[x, y] \leftarrow \text{NA}
\}
\]
\[
\text{if} (y == \text{length(topArticlesTrain90tw)})
\]
\[
\text{y} = 1
\]
\[
\text{else} \{
\text{y} = y + 1
\}
\]
\[
\# \text{end for}
\text{x} = x + 1
\]
\[
\# \text{end for}
\]
\[
\text{endTime} \leftarrow \text{as.numeric(Sys.time())} \# \text{endOf dissimilarity Mat}
\]

#### Make avgDistanceTw, stdDistance and avgDistanceTwWeighted

# Average distance and std dev

####
# need to ignore rows of NAs
\[
\text{validRow} \leftarrow 1
\]
\[
\text{while}(\text{length(\text{which}(\text{is.na(classificationMatrixTw[validRow,]))}) == ncol(classificationMatrixTw))) \{ \# \text{every value is NA, invalid row}
\text{validRow} \leftarrow \text{validRow} + 1 \# \text{find first valid row}
\}
\]
\[
\text{invalidRows} \leftarrow \text{which(\text{is.na(classificationMatrixTw[,1]))} \# \text{rows of all NA}
\text{toRemove} \leftarrow \text{which(\text{is.na(classificationMatrixTw[validRow,]))} \# \text{NA columns}
\text{(not sure why - mostly erroneous or too short)}
\]

# Get weightsTw
\[
x<-1
\]
\[
\text{weightsTw} \leftarrow \text{vector(length = length(topArticlesTrain90tw), mode = "numeric")}
\]
\[
\text{for}(i \in \text{topArticlesTrain90tw}) \{ \text{weightsTw}[x] \leftarrow \text{data[[i]]$mostRecentTwShares; x} <- x + 1 \} \# \text{make vector of final twShares}
\]

# Make avg, sd, wavg vectors
\[
\text{if}(\text{length(toRemove)} == 0) \{
\text{avgDistanceTw} \leftarrow \text{rowMeans(classificationMatrixTw)}
\text{stdDistance} \leftarrow \text{apply(classificationMatrixTw, 1, sd)}
\text{weightedavgDistanceTw} \leftarrow \text{apply(classificationMatrixTw, 1, weighted.mean, w = weightsTw, na.rm = TRUE)}
\}
\]
\[
\text{else} \{
\text{avgDistanceTw} \leftarrow \text{rowMeans(classificationMatrixTw[-toRemove])} \# \text{average distance for each row (test Article) to all train articles}
\text{stdDistance} \leftarrow \text{apply(classificationMatrixTw[-toRemove], 1, sd)}
\text{weightedavgDistanceTw} \leftarrow \text{apply(classificationMatrixTw[-toRemove], 1, weighted.mean, w = weightsTw[-toRemove], na.rm = TRUE)}
\}
\]

#### Make Classification reference Matrix
# create classification matrix

```r
classificationReferenceTw <- matrix(nrow = length(randomTestArticles), ncol = 10)
colnames(classificationReferenceTw) <- c("id", "url", "dataRef", "viral10", "viral20", "averageDist", "weightedDistance", "sdDist", "30MinCheck", "timeDiffCheck")
x<-1
for (article in randomTestArticles) { # x values in classificationMatrixTw3 correspond to RandomTestArticles
    classificationReferenceTw[x, "id"] <- data[[article]]$id
    classificationReferenceTw[x, "url"] <- data[[article]]$sourceUrl
    classificationReferenceTw[x, "dataRef"] <- article
    if (data[[article]]$mostRecentTwShares >= cutoff90tw){
        classificationReferenceTw[x, "viral10"] <- 1
    } else {classificationReferenceTw[x, "viral10"] <- 0}
    if (data[[article]]$mostRecentTwShares >= cutoff80tw){
        classificationReferenceTw[x, "viral20"] <- 1
    } else {
        classificationReferenceTw[x, "viral20"] <- 0
    }classificationReferenceTw[x, "averageDist"] <- as.numeric(avgDistanceTw[x])
classificationReferenceTw[x, "sdDist"] <- as.numeric(stdDistance[x])
classificationReferenceTw[x, "weightedDistance"] <- as.numeric(weightedavgDistanceTw[x])
    if (nrow(data[[article]]$mergedMat) >= data[[article]]$twFirstActivity+timeDiff ) { # is 15 ahead to look
        classificationReferenceTw[x, "30MinCheck"] <- data[[article]]$mergedMat[data[[article]]$twFirstActivity+10,"twShares"]
        classificationReferenceTw[x, "timeDiffCheck"] <- data[[article]]$mergedMat[data[[article]]$twFirstActivity+15,"twShares"]
    } else { if(nrow(data[[article]]$mergedMat) >= data[[article]]$twFirstActivity+10) { # is 10 ahead to look
        classificationReferenceTw[x, "30MinCheck"] <- data[[article]]$mergedMat[data[[article]]$twFirstActivity+10,"twShares"]
        classificationReferenceTw[x, "timeDiffCheck"] <- data[[article]]$mostRecentTwShares
    } else { # first activity is <30 mins away from end obs
        classificationReferenceTw[x, "30MinCheck"] <- data[[article]]$mostRecentTwShares
        classificationReferenceTw[x, "timeDiffCheck"] <- data[[article]]$mostRecentTwShares
    }
}
```

Compare viral classifications for the nominal and DTW models
# 11a - Model Comparison Matrix Twitter
# Refs at 10,15,20,30,40,50
# Train sample refs at 50,100,...,300

classificationReferenceTw = classificationReferenceTwTrain300
timeDiff= 10
# change: modelComparisonMatrixTw,
# Compare, vanilla model to weighted average to reality
# rank Test articles all the way through...

modelComparisonMatrixTw <- matrix(nrow = length(randomTestArticles), ncol = 7)
colnames(modelComparisonMatrixTw) <- c("Article", "avgDist", "avgLevel", "wAvgDist", "nomCount", "finalCount", "viral10")
modelComparisonMatrixTw[,"Article"] <- as.numeric(randomTestArticles)
modelComparisonMatrixTw[,"avgDist"] <- as.numeric(avgDistanceTw)
modelComparisonMatrixTw[,"wAvgDist"] <- as.numeric(classificationReferenceTw[, "weightedDistance"])
modelComparisonMatrixTw[,"nomCount"] <- as.numeric(classificationReferenceTw[, "30MinCheck"])

for( i in 1:length(randomTestArticles)){
  modelComparisonMatrixTw[i,"finalCount"] <- as.numeric(data[[randomTestArticles[i]]]$mostRecentTwShares)
}
modelComparisonMatrixTw[,"avgLevel"] <- NA

as.numeric(classificationVectorTw)
modelComparisonMatrixTw[,"viral10"] <- classificationReferenceTw[, "viral10"]
class(modelComparisonMatrixTw) <- "numeric"
# View(modelComparisonMatrixTw)
# extract ordered articles from each of the 4 methods and sort

sortedByAvgTw <- modelComparisonMatrixTw[sort.list(modelComparisonMatrixTw[, "avgDist"], decreasing = FALSE), ] # low is good
sortedByWavgTw <- modelComparisonMatrixTw[sort.list(modelComparisonMatrixTw[, "wAvgDist"], decreasing = FALSE), ] # low is good
sortedByNomTw <- modelComparisonMatrixTw[sort.list(modelComparisonMatrixTw[, "nomCount"], decreasing=TRUE), ] # high is good
sortedByFinalTw <- modelComparisonMatrixTw[sort.list(modelComparisonMatrixTw[, "finalCount"], decreasing=TRUE), ] # high is good
sortedByLevelTw <- modelComparisonMatrixTw[sort.list(modelComparisonMatrixTw[, "avgLevel"], decreasing = FALSE), ] # low is good

sortedClassificationsTw <- matrix(nrow = length(randomTestArticles), ncol = 5)
colnames(sortedClassificationsTw) <- c("avgDist","avgLevel", "wAvgDist", "nomCount", "finalCount")

sortedClassificationsTw[, "avgDist"] <- modelComparisonMatrixTw[sort.list(modelComparisonMatrixTw[, "avgDist"]), "Article"]
sortedClassificationsTw[, "wAvgDist"] <- modelComparisonMatrixTw[sort.list(modelComparisonMatrixTw[, "wAvgDist"]), "Article"]
sortedClassificationsTw[, "nomCount"] <- modelComparisonMatrixTw[sort.list(modelComparisonMatrixTw[, "nomCount"], decreasing=TRUE), "Article"
sortedClassificationsTw[, "finalCount"] <- modelComparisonMatrixTw[sort.list(modelComparisonMatrixTw[, "finalCount"], decreasing=TRUE), "Article"
sortedClassificationsTw[, "avgLevel"] <- modelComparisonMatrixTw[sort.list(modelComparisonMatrixTw[, "avgLevel"], decreasing=FALSE), "Article"

# as comparison, T/F values for viral or not
# examine side by side
sortedClassificationsViralTw <- matrix(nrow = length(randomTestArticles), ncol = 5)
colnames(sortedClassificationsViralTw) <- c("avgDist", "avgLevel", "wAvgDist", "nomCount", "finalCount")

maxNumProposed <- 350 # The number of articles to be classified viral
wAvgUniqueAcc <- vector(length = maxNumProposed, mode="numeric") # vector to store unique and correct article
wAvgUniqueNum <- vector(length = maxNumProposed, mode="numeric") # vector to store number of unique articles proposed
modelComparisonAccuracy <- matrix(nrow = maxNumProposed, ncol = 6)
colnames(modelComparisonAccuracy) <- c("proposed", "avgAcc", "levelAcc", "wavgAcc", "nomAcc", "overlap")

for (numProposed in 1:maxNumProposed){
  modelComparisonAccuracy[numProposed, "proposed"] <- numProposed
  modelComparisonAccuracy[numProposed, "avgAcc"] <- length(which(sortedByAvgTw[1:numProposed, "viral10"]==1))
  modelComparisonAccuracy[numProposed, "levelAcc"] <- length(which(sortedByLevelTw[1:numProposed, "viral10"]==1))

  modelComparisonAccuracy[numProposed, "wavgAcc"] <- length(which(sortedByWavgTw[1:numProposed, "viral10"]==1))
  modelComparisonAccuracy[numProposed, "nomAcc"] <- length(which(sortedByNomTw[1:numProposed, "viral10"]==1))

  modelComparisonAccuracy[numProposed, "overlap"] <- length(which(sortedByWavgTw[which(sortedByWavgTw[1:numProposed, "viral10"]==1), "Article"] %in% sortedByNomTw[which(sortedByNomTw[1:numProposed, "viral10"]==1), "Article"]))
}

# also calculate wAvg unique accuracy here
uniqueViral=0
for ( i in differingArticles){
  if (data[[i]]$mostRecentTwShare >= cutoff90tw){
    uniqueViral <- uniqueViral +1
  } else {
    wAvgUniqueAcc[numProposed] <-NA
    wAvgUniqueNum[numProposed] <- length(differingArticles)
  }
}

D.14

Analysis of various cutoff points and tradeoffs

# 12a - cutoff analysis

classificationReferenceTw = classificationReferenceTw40
timeDiff = 40

# make predictions

---

```r
write.csv(modelComparisonAccuracy, row.names=TRUE, 
           file=paste("modelComparisonAccuracyDTW", timeDiff, 
                      
           aucWavg <- auc(modelComparisonAccuracy[, "wavgAcc"], 
                       y = c(1:nrow(modelComparisonAccuracy)))

aucNom <- auc(modelComparisonAccuracy[, "nomAcc"], 
                y = c(1:nrow(modelComparisonAccuracy)))

graphLineWidth <- 2.5

plot(modelComparisonAccuracy[, "proposed"], type="l", col = "black", 
     main = paste("Model Comparison Graph\n\nTime Diff = ", timeDiff*3, 
     xlab = "Number Proposed", ylab = "Number Viral", 
     lwd = graphLineWidth, 
     lwd = graphLineWidth)

text(x=300, y=225, labels=paste("wAvg AUC = ", aucWavg, "\nNom AUC = ", aucNom))

abline(h = length(which(classificationReferenceTw[, "viral10"] == 1)), col = "grey")

lines(modelComparisonAccuracy[, "avgAcc"], type='l', col = "blue", lwd = graphLineWidth)

lines(modelComparisonAccuracy[, "wavgAcc"], type='l', col = "red", lwd = graphLineWidth)

lines(modelComparisonAccuracy[, "nomAcc"], type='l', col = "green", lwd = graphLineWidth)

lines(modelComparisonAccuracy[, "overlap"], type='l', col = "plum2", lwd = graphLineWidth)

lines(modelComparisonAccuracy[, "levelAcc"], type='l', col = "orange", lwd = graphLineWidth)

lines(wAvgUniqueAcc, type = 'l', col = "darkorchid3", lwd = graphLineWidth)

legendText = c("proposed", "avgAcc", "wavgAcc", "nomAcc", "overlap", "wAvg uniAcc", "levelAcc")

legendCol = c("black", "blue", "red", "green", "plum2", "darkorchid3", 
               "orange")

legend(x="topleft", legend = legendText, col = legendCol, lwd = graphLineWidth)

#plot(wAvgUniqueAcc/wAvgUniqueNum, type='l', 
#     main= paste("Accuracy of Unique Predictions\n\nTime Diff = ", 
#     h = 0.1, col="grey")

write.csv(wAvgUniqueAcc/wAvgUniqueNum, file = paste("Unique Accuracy",timeDiff,".csv"))
```

```
Analysis of various cutoff points and tradeoffs

# 12a - cutoff analysis

classificationReferenceTw = classificationReferenceTw40
timeDiff = 40

# make predictions
```
unweightedCutoffTw <- 53
weightedCutoffTw <- 24

weightedSequence <- seq(from = 600, to = 950, by = 20)
tradeoffMatrixTw <- matrix(nrow = length(weightedSequence), ncol = 9)
colnames(tradeoffMatrixTw) <- c("cutoffDist", "accuracy", "positiveClass", "truePos", "falsePos", "trueNeg", "falseNeg", "recAcc", "pickupRate")

position <- 1

# Make predictions
for(weightedCutoffTw in weightedSequence){
  viral10VectorTw <- which(classificationReferenceTw[,"viral10"] == 1)
  viral20VectorTw <- which(classificationReferenceTw[,"viral20"] == 1)
  predictedVectorUnweightedTw <- which(classificationReferenceTw[,"averageDist"] <= unweightedCutoffTw) # predictedVectorUnweightedTw
  predictedVectorWeightedTw <- which(classificationReferenceTw[,"weightedDistance"] <= weightedCutoffTw) # predictedVectorWeightedTw
  predictedCombo <- unique(c(predictedVectorUnweightedTw, predictedVectorWeightedTw))
}

### Measure accuracy and pickup rates

# get accuracy and pickup of each of the two types of check (and combo) 
#(might be worth looking at again, not sure if these are put out with the classification tables)

recSuccessUnweighted <- length(which(viral10VectorTw %in% predictedVectorUnweightedTw))/length(predictedVectorUnweightedTw) # unweighted distances - 74% OR 100% at 1100
recPickupUnweighted <- length(which(viral10VectorTw %in% predictedVectorUnweightedTw))/length(viral10VectorTw)

recSuccessWeighted <- length(which(viral10VectorTw %in% predictedVectorWeightedTw))/length(predictedVectorWeightedTw) # weighted dist 85% OR over 95% at 2100
recPickupWeighted <- length(which(viral10VectorTw %in% predictedVectorWeightedTw))/length(viral10VectorTw) # weighted dist 85% OR over 95% at 2100

recSuccessCombo <- length(which(viral10VectorTw %in% predictedCombo))/length(predictedCombo) # combined results - 80%
recPickupCombo <- length(which(viral10VectorTw %in% predictedCombo))/length(viral10VectorTw)

overlapPredictions <- length(which(predictedVectorUnweightedTw %in% predictedVectorWeightedTw)) # pred==YES articles in both lists
overlapSuccess <- overlapPredictions/length(viral10VectorTw)

# get weighted, unweighted, combo and actual classifications


predictViralUnweighted <- vector(length = length(randomTestArticles), mode="numeric")
for (i in predictedVectorUnweightedTw){  # cycle through unweighted predictions of viral==YES
    predictViralUnweighted[i] <- 1
}
predictViralWeighted <- vector(length = length(randomTestArticles), mode="numeric")
for (i in predictedVectorWeightedTw){  # cycle through unweighted predictions of viral==YES
    predictViralWeighted[i] <- 1
}
predictViralCombo <- vector(length = length(randomTestArticles), mode="numeric")
for (i in predictedCombo){  # cycle through unweighted predictions of viral==YES
    predictViralCombo[i] <- 1
}
actualViral <- vector(length = length(randomTestArticles), mode="numeric")
for (i in viral10VectorTw){
    actualViral[i] <- 1
}

# Weighted confusion matrix
truePosWeighted <- 0
falsePosWeighted <- 0
tureNegWeighted <- 0
falseNegWeighted <- 0
error <- 0
for (i in 1:length(predictViralWeighted)){
    if(predictViralWeighted[i] == 1 & & actualViral[i] == 1){# TP - Actual = yes, pred = yes
        truePosWeighted <- truePosWeighted+1
    } else if(predictViralWeighted[i] == 1 & & actualViral[i] == 0){ # FP - pred = yes, actual = no
        falsePosWeighted <- falsePosWeighted+1
    } else if(predictViralWeighted[i] == 0 & & actualViral[i] == 0){ # TN pred = no, actual = no
        trueNegWeighted <- trueNegWeighted+1
    } else if(predictViralWeighted[i] == 0 & & actualViral[i] == 1){ # FN pred = no, actual = yes
        falseNegWeighted <- falseNegWeighted+1
    } else {error <- error + 1}
}
truePosRateWeighted <- truePosWeighted/(truePosWeighted+falseNegWeighted)  # TP/(TP+FN)
falsePosRateWeighted <- falsePosWeighted/(falsePosWeighted + truePosWeighted)  # changed the formula, this number is more meaningful
trueNegRateWeighted <- trueNegWeighted/(trueNegWeighted+ falsePosWeighted)
accuracyWeighted <- (truePosWeighted + trueNegWeighted)/(length(predictViralWeighted))#tp+tn/p+n

# Output confusion matrix - Weighted
cat("\t\tWeighted \n Cutoff = ", weightedCutoffTw,"\n")
### # make tradeoff comparison matrix

```
# D.17
#
# Make tradeoff comparison matrix

trdoffMatrTw[|position, "cutoffDist"] <- weightedCutoffTw
trdoffMatrTw[|position, "accuracy"] <- accuracyWeighted
trdoffMatrTw[|position, "positiveClass"] <- length(which(predictViralWeighted==1)
trdoffMatrTw[|position, "truePos"] <- truePosWeighted
trdoffMatrTw[|position, "falsePos"] <- falsePosWeighted
trdoffMatrTw[|position, "trueNeg"] <- trueNegWeighted
trdoffMatrTw[|position, "falseNeg"] <- falseNegWeighted
trdoffMatrTw[|position, "recAcc"] <- trdoffMatrTw[|position, "truePos"] / trdoffMatrTw[|position, "positiveClass"]
trdoffMatrTw[|position, "pickupRate"] <- truePosWeighted/length(which(actualViral==1))
```

```l
position <- position + 1
```

```
plot(trdoffMatrTw[, "recAcc"], main = 'Pickup Rate vs Rec Accuracy',
ylab = "Percentage %", xlab = "Cutoff Distance", xaxt = 'n',
   type='l', col = "blue", lwd = 2)
```

```
axis(side = 1, labels = trdoffMatrTw[, "cutoffDist"], at =
c(1:nrow(trdoffMatrTw)))
lines(trdoffMatrTw[, "pickupRate"], type='l', col='red', lwd = 2)
legend(x="right", lwd = 2,
   legend=c("Pickup Rate", "Rec Accuracy"),
   col=c("Red", "blue"))
```

```
write.csv(file="tradeoffMatrix30Mins.csv", trdoffMatrTv)
```

```r
# Get AUC for the two lines of interest

cAucRecAcc <- auc(trdoffMatrTv[, "recAcc"], y=c(1:nrow(trdoffMatrTv)))
cAucPickup <- auc(trdoffMatrTv[, "pickupRate"], y=c(1:nrow(trdoffMatrTv)))
```

```
cor(trdoffMatrTv[, "recAcc"], trdoffMatrTv[, "pickupRate"])
```

```
# boxplot of distances for viral and not

boxplot(as.numeric(classificationReferenceTw[which(classificationReferenceTw[,"viral10"] == 1), "weightedDistance" )],
as.numeric(classificationReferenceTw[which(classificationReferenceTw[,"viral10"] == 0), "weightedDistance" ]),
  pars=list(ylim=c(550,1100)), notc = TRUE, xlab="Viral?", main =
  "Distance Boxplots Tw",
  col = c("green3", "red2"), horizontal=F) # Boxplot of viral article widths
```
# Combo confusion matrix

```r
truePosCombo <- 0
falsePosCombo <- 0
trueNegCombo <- 0
falseNegCombo <- 0
for (i in 1:length(predictViralCombo)){
  if(predictViralCombo[i] == 1 & & actualViral[i] ==1){# TP - Actual = yes, predict = yes
    truePosCombo <- truePosCombo+1
  } else if(predictViralCombo[i] == 1 & & actualViral[i] ==0){# FP - predict = yes, actual = no
    falsePosCombo <- falsePosCombo+1
  } else if(predictViralCombo[i] ==0 & & actualViral[i] ==0){# TN pred = no, actual = no
    trueNegCombo <- trueNegCombo+1
  } else if(predictViralCombo[i] ==0 & & actualViral[i] ==1){# FN pred=no, actual = yes
    falseNegCombo <- falseNegCombo+1
  } else {
    error <- error + 1
  }
}
```

```r
truePosRateCombo <- truePosCombo/(truePosCombo+falseNegCombo)# TP/(TP+FN)
falsePosRateCombo <- falsePosCombo/(falsePosCombo + truePosCombo) # changed the formula, this number is more meaningful
trueNegRateCombo <- trueNegCombo/(trueNegCombo+ falsePosCombo)
accuracyCombo <- (truePosCombo + trueNegCombo)/length(predictViralCombo)#tp+tn/p+n
```

```r
# Output confusion matrix - Combo
cat("Actual\n1 0\nPred\t1 0\n\nTrue Pos\t", truePosCombo, "\nFalse Pos\t", falsePosCombo,"\nTrue Neg\t", trueNegCombo, "\n\nAccuracy = ",accuracyCombo,"\n\n\n\n\n```

```r
# Output confusion matrix - Weighted
cat("\n\n\n\n```

```r
axis(side = 1, labels=c("Viral", "Non-Viral"), at = c(1:2))
```

```r
truePosWeighted <- truePosCombo/1000
falsePosWeighted <- falsePosCombo/1000
trueNegWeighted <- trueNegCombo/1000
falseNegWeighted <- falseNegCombo/1000
```

```r
# Output confusion matrix - Combo
cat("\n\n\n```

```r
# Output confusion matrix - Weighted
cat("\n\n\n```

```r
accuracyWeighted <- truePosWeighted/1000
```

```r
# Combo confusion matrix
```
### EndOf Weighted confusion matrix

### Unweighted confusion matrix

```r
truePosUnweighted <- 0
falsePosUnweighted <- 0
trueNegUnweighted <- 0
falseNegUnweighted <- 0
error <- 0
for (i in 1:length(predictViralUnweighted)){
  if(predictViralUnweighted[i] == 1 & & actualViral[i] == 1){ # TP - Actual = yes, predict = yes
    truePosUnweighted <- truePosUnweighted+1
  } else if(predictViralUnweighted[i] == 1 & & actualViral[i] == 0){ # FP - predic = yes, actual = no
    falsePosUnweighted <- falsePosUnweighted+1
  } else if(predictViralUnweighted[i] == 0 & & actualViral[i] == 0){ # TN pred = no, actual = no
    trueNegUnweighted <- trueNegUnweighted+1
  } else if(predictViralUnweighted[i] == 0 & & actualViral[i] == 1){ # FN pred=no, actual = yes
    falseNegUnweighted <- falseNegUnweighted+1
  } else {
    error <- error + 1
  }
}
truePosRateUnweighted <- truePosUnweighted/(truePosUnweighted+falseNegUnweighted) # TP/(TP+FN)
falsePosRateUnweighted <- falsePosUnweighted/(falsePosUnweighted + truePosUnweighted) # changed the formula, this number is more meaningful
trueNegRateUnweighted <- trueNegUnweighted/(trueNegUnweighted+falsePosUnweighted)
accuracyUnweighted <- (truePosUnweighted + trueNegUnweighted)/(length(predictViralUnweighted)) #tp+tn/p+n

# Output confusion matrix - Unweighted
cat("\t\tUnweighted \n Cutoff = ", unweightedCutoffTw, "\n",
  "\t\tACTUAL\n\t 1 \t 0\n Pred\tl1\t", truePosUnweighted, "\t",
falsePosUnweighted,
  " \t0\t",falseNegUnweighted," \t", trueNegUnweighted, " \n\n Accuracy = ",accuracyUnweighted,"\n \tActual Viral - ", truePosUnweighted + falseNegUnweighted, "
 \tViral mised (FN) - ", falseNegUnweighted,
 \tBad Recommendations (FP) - ", falsePosUnweighted,"
 \tArticels recomended - ", falsePosUnweighted+truePosUnweighted,
 "\t\ttruePosUnweighted/(falsePosUnweighted+truePosUnweighted),"% good), ", "
 \tTPR - ", truePosRateUnweighted, "%"")

### EndOf Unweighted confusion matrix
```

### Output activity for all publishers

#### E - Most active Publishers

# Find the number of facebook and twitter shares that each publisher gets in these sets
# (very small, use larger example also: http://gigaom.com/2014/02/03/facebook-and-the-race-to-make-content-go-viral-meet-the-new-boss-same-as-the-old-boss/)
# make publisher variable for articles
# take info up to the 3rd '/' in $sourceUrl
publisherVector <- vector(length = length(length(data2)), mode = "character")
for (i in 1:length(data2)){
  # get str position of third /
  slashPosition3 <- as.numeric(gregexpr("/", data2[[i]]$sourceUrl)[[1]][3])
  data2[[i]]$publisher <- substr(data2[[i]]$sourceUrl, 1, slashPosition3)
  publisherVector[i] <- data2[[i]]$publisher
cat(" - ", i)
}
# count number of stories published
uniquePublisherVector <- unique(publisherVector) # 1581 unique publishers

publisherMatrix <- matrix(nrow = length(uniquePublisherVector), ncol = 5)
colnames(publisherMatrix) <- c("articleCount", "twShares", "fbShares", "fbLikes", "fbComms")
publisherMatrix[,1:5] <- 0
for (i in 1:length(data2)){
  index <- which(uniquePublisherVector == data2[[i]]$publisher)
  publisherMatrix[index,"articleCount"] <- publisherMatrix[index,"articleCount"] + 1
  publisherMatrix[index,"twShares"] <- publisherMatrix[index,"twShares"] + data2[[i]]$mostRecentTwShares
  publisherMatrix[index,"fbShares"] <- publisherMatrix[index,"fbShares"] + data2[[i]]$mostRecentFbShares
  publisherMatrix[index,"fbLikes"] <- publisherMatrix[index,"fbLikes"] + data2[[i]]$mostRecentFbLikes
  publisherMatrix[index,"fbComms"] <- publisherMatrix[index,"fbComms"] + data2[[i]]$mostRecentFbComments
}
View(publisherMatrix)
write.csv(publisherMatrix, file = "publisherMatrix.csv")
write.csv(uniquePublisherVector, file = "publisherVector.csv")

# average category series
# NB: 291 and 292 are missing as category references
# step 1, convert all the categories from text to numbers
for (i in 1:length(data2)){
  data[[i]]$categories <- as.numeric(data[[i]]$categories)
}
# step 2, get all the categories
catMatrix <- matrix(nrow = nrow(nwCategories), ncol = 8)
```r
colnames(catMatrix) <- c("categoryName","category","articleCount","articleCountNoSM","twShares","fbShares","fbLikes","fbComms")
catMatrix[,"category"] <- nwCategories[,1]
catMatrix[,"categoryName"] <- 0
catMatrix[,2:8] <- 0

# step 3: cycle through each article, add to the counter
for (i in 1:length(data)){
  for (cat in 1:length(data[[i]]$categories)){
    thisCat <- which(catMatrix[,"category"] == data[[i]]$categories[cat])
    catMatrix[thisCat,"articleCount"] <- catMatrix[thisCat,"articleCount"] +1
    catMatrix[thisCat,"twShares"] <- catMatrix[thisCat,"twShares"] +data[[i]]$mostRecentTwShares
    catMatrix[thisCat,"fbShares"] <- catMatrix[thisCat,"fbShares"] +data[[i]]$mostRecentFbShares
    catMatrix[thisCat,"fbLikes"] <- catMatrix[thisCat,"fbLikes"] +data[[i]]$mostRecentFbLikes
    catMatrix[thisCat,"fbComms"] <- catMatrix[thisCat,"fbComms"] +data[[i]]$mostRecentFbComments
  }
}

### Count number of articles from each category with no social Media Presence
for (i in 1:length(data2)){
  data2[[i]]$categories <- as.numeric(data2[[i]]$categories)
}

for (i in 1:length(data2)){
  for (cat in 1:length(data2[[i]]$categories)){
    thisCat <- which(catMatrix[,"category"] == data2[[i]]$categories[cat])
    catMatrix[thisCat,"articleCountNoSM"] <- catMatrix[thisCat,"articleCountNoSM"] +1
  }
}
View(catMatrix)
write.csv(catMatrix, file = "categoryMatrixTotal.csv")
```
### E. R Packages Used

One of R’s greatest strengths is its large and varied repository of functions and packages. The packages used are outlined in this Appendix in chronological order.

<table>
<thead>
<tr>
<th>Package</th>
<th>rjson</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release</td>
<td>0.2.12</td>
</tr>
<tr>
<td>Year</td>
<td>2012</td>
</tr>
<tr>
<td>Contact</td>
<td>Alex Couture-Beil <a href="mailto:rjson_pkg@mofo.ca">rjson_pkg@mofo.ca</a></td>
</tr>
<tr>
<td>R Version</td>
<td>&gt;= 2.12.0</td>
</tr>
<tr>
<td>Description</td>
<td>Rjson is a package used to interact with data sets of the JSON type and to write sets of that type.</td>
</tr>
<tr>
<td>Functions Used</td>
<td>fromJSON</td>
</tr>
<tr>
<td>Nature of use</td>
<td>Interpret the JSON output from NewsWhip's servers.</td>
</tr>
<tr>
<td>Reference</td>
<td>Couture-Beil (2013)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Package</th>
<th>Flux</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release</td>
<td>0.2-2</td>
</tr>
<tr>
<td>Year</td>
<td>2013</td>
</tr>
<tr>
<td>Contact</td>
<td>Gerald Jurasinski <a href="mailto:gerald.jurasinski@unirostock.de">gerald.jurasinski@unirostock.de</a></td>
</tr>
<tr>
<td>R Version</td>
<td>&gt;= 2.12.0</td>
</tr>
<tr>
<td>Description</td>
<td>Contains functions for the calculation of greenhouse gas flux rates in closed chamber experiments.</td>
</tr>
<tr>
<td>Functions Used</td>
<td>auc</td>
</tr>
<tr>
<td>Nature of use</td>
<td>Determining the area under a curve for model comparison</td>
</tr>
<tr>
<td>Reference</td>
<td>Jurasinski et al. (2013)</td>
</tr>
<tr>
<td>Package</td>
<td>zoo</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>Release</td>
<td>1.7-11</td>
</tr>
<tr>
<td>Year</td>
<td>2005</td>
</tr>
<tr>
<td>Contact</td>
<td>Achim Zeileis  <a href="mailto:Achim.Zeileis@R-project.org">Achim.Zeileis@R-project.org</a></td>
</tr>
<tr>
<td>R Version</td>
<td>&gt;= 2.10.0</td>
</tr>
<tr>
<td>Description</td>
<td>An S3 class with methods for ordered indexed observations.</td>
</tr>
<tr>
<td>Functions Used</td>
<td>na.approx</td>
</tr>
<tr>
<td>Nature of use</td>
<td>Interpolation values as per formula outlined in Section 3.2.</td>
</tr>
</tbody>
</table>
This section evaluates all the variations on the models on several criteria. Note that these models have been run on various splits of training.

1. Model Accuracy (relative to Nominal Model)
2. Unique Accuracy
3. Pickup Rate vs. True Positive Rate
4. Calculation time

In this appendix, detailed comparisons of model accuracy will be made using several metrics, these are; pickup rate (sensitivity), specificity, recommendation accuracy (precision), Receiver Operating Characteristic (ROC) curves, Area Under the Curve (AUC) and regression.

*Sensitivity* refers to the proportion of actually viral articles that the model has picked up. It is often referred to as the True Positive Rate (TPR) and follows the function:

\[
TPR = \frac{TP}{P} = \frac{TP}{TP + FN}
\]

*Specificity* is the True Negative Rate. It measures the ability to exclude non-negative articles. It can be expressed as:

\[
SPC = TNR = \frac{TN}{N} = \frac{TN}{FP + TN}
\]

*Precision (or Positive Predictive Value)* is the proportion of the positively classified articles that are correctly classified. In the context of this model, it is the number of articles proposed as viral that ultimately turn out to be. It is sometimes referred to as Recommendation Accuracy and is calculated by:

\[
PPV = \frac{TP}{TP + FP}
\]

*Receiver Operating Characteristic (ROC) curve* is the plot of the True Positive Rate against the False Positive Rate \((1 - specificity)\) for various cutoff points.

*Area Under the Curve (AUC)* is the simplest single method of comparing models. The greater the AUC for the accuracy or ROC curve of a model, the better it is. The model with the highest AUC can be said to be generally superior. AUC is the integral of a line according to the function:

\[
Area = \int_a^b f(x) \, dx
\]

An AUC of the ROC curve is generally considered to be the primary indicator of model efficacy. An ROC AUC of 0.5 means the model’s ability to classify between viral and not is no better than if done by chance. A value of 1 is a perfect prediction model.

Throughout, Regression analysis is used to derive general formulas for how the models perform. Regression is used to predict the relationship between variables, for example, the
number of articles proposed and the number of articles correctly classified. Linear, Exponential, Logarithmic and Polynomial regression lines are all used as appropriate. The \( R^2 \) values will also be given where appropriate. This value represents the goodness of fit of a model. It is the linear dissimilarity of the points to the line drawn and exists between 0 and 1 with 1 being a perfectly fit model.

Each of these will be exemplified using predictions made at 30 minutes after the first activity in Dataset B. The raw output for this model at various cutoffs can be seen in Table F.1

Figure F.1 compares the models in operation. Each model is represented by a single line and has been requested to propose 350 articles as potentially viral. All the train articles were incorporated and the classifications were performed 30 minutes after first activity.

As is visible above, the existing Nominal model outperforms the Weighted Average DTW Distance model. Examining the Area Under the Curve (AUC) for these models indicates the extent to which the models differ. The nominal model has 19,653 while the DTW model has only 18,315. We can also use this plot to derive an equation for how many articles a model will

<table>
<thead>
<tr>
<th>Cutoff Distance</th>
<th>Accuracy</th>
<th>Positive Class</th>
<th>truePos</th>
<th>falsePos</th>
<th>trueNeg</th>
<th>falseNeg</th>
<th>Sensitivity (TPR) (Pickup)</th>
<th>FPR</th>
<th>Specificity (TNR)</th>
<th>Precision (RecAcc)</th>
</tr>
</thead>
<tbody>
<tr>
<td>600</td>
<td>0.902</td>
<td>22</td>
<td>22</td>
<td>0</td>
<td>1685</td>
<td>185</td>
<td>0.106</td>
<td>0.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>620</td>
<td>0.904</td>
<td>26</td>
<td>26</td>
<td>0</td>
<td>1685</td>
<td>181</td>
<td>0.126</td>
<td>0.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>640</td>
<td>0.912</td>
<td>40</td>
<td>40</td>
<td>0</td>
<td>1685</td>
<td>167</td>
<td>0.193</td>
<td>0.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>660</td>
<td>0.919</td>
<td>53</td>
<td>53</td>
<td>0</td>
<td>1685</td>
<td>154</td>
<td>0.256</td>
<td>0.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>680</td>
<td>0.929</td>
<td>79</td>
<td>76</td>
<td>3</td>
<td>1682</td>
<td>131</td>
<td>0.367</td>
<td>0.002</td>
<td>0.998</td>
<td>0.962</td>
</tr>
<tr>
<td>700</td>
<td>0.934</td>
<td>103</td>
<td>93</td>
<td>10</td>
<td>1675</td>
<td>114</td>
<td>0.449</td>
<td>0.006</td>
<td>0.994</td>
<td>0.903</td>
</tr>
<tr>
<td>720</td>
<td>0.941</td>
<td>126</td>
<td>111</td>
<td>15</td>
<td>1670</td>
<td>96</td>
<td>0.536</td>
<td>0.009</td>
<td>0.991</td>
<td>0.881</td>
</tr>
<tr>
<td>740</td>
<td>0.936</td>
<td>158</td>
<td>122</td>
<td>36</td>
<td>1649</td>
<td>85</td>
<td>0.589</td>
<td>0.021</td>
<td>0.979</td>
<td>0.772</td>
</tr>
<tr>
<td>760</td>
<td>0.931</td>
<td>202</td>
<td>139</td>
<td>63</td>
<td>1622</td>
<td>68</td>
<td>0.671</td>
<td>0.037</td>
<td>0.963</td>
<td>0.688</td>
</tr>
<tr>
<td>780</td>
<td>0.918</td>
<td>252</td>
<td>152</td>
<td>100</td>
<td>1585</td>
<td>55</td>
<td>0.734</td>
<td>0.059</td>
<td>0.941</td>
<td>0.603</td>
</tr>
<tr>
<td>800</td>
<td>0.900</td>
<td>305</td>
<td>161</td>
<td>144</td>
<td>1541</td>
<td>46</td>
<td>0.778</td>
<td>0.085</td>
<td>0.915</td>
<td>0.528</td>
</tr>
<tr>
<td>820</td>
<td>0.875</td>
<td>368</td>
<td>169</td>
<td>199</td>
<td>1486</td>
<td>38</td>
<td>0.816</td>
<td>0.118</td>
<td>0.882</td>
<td>0.459</td>
</tr>
<tr>
<td>840</td>
<td>0.841</td>
<td>446</td>
<td>176</td>
<td>270</td>
<td>1415</td>
<td>31</td>
<td>0.850</td>
<td>0.160</td>
<td>0.840</td>
<td>0.395</td>
</tr>
<tr>
<td>860</td>
<td>0.774</td>
<td>580</td>
<td>180</td>
<td>400</td>
<td>1285</td>
<td>27</td>
<td>0.870</td>
<td>0.237</td>
<td>0.763</td>
<td>0.310</td>
</tr>
<tr>
<td>880</td>
<td>0.673</td>
<td>796</td>
<td>192</td>
<td>604</td>
<td>1081</td>
<td>15</td>
<td>0.928</td>
<td>0.358</td>
<td>0.642</td>
<td>0.241</td>
</tr>
<tr>
<td>900</td>
<td>0.498</td>
<td>1145</td>
<td>201</td>
<td>944</td>
<td>741</td>
<td>6</td>
<td>0.971</td>
<td>0.560</td>
<td>0.440</td>
<td>0.176</td>
</tr>
<tr>
<td>920</td>
<td>0.217</td>
<td>1684</td>
<td>205</td>
<td>1479</td>
<td>206</td>
<td>2</td>
<td>0.990</td>
<td>0.878</td>
<td>0.122</td>
<td>0.122</td>
</tr>
<tr>
<td>940</td>
<td>0.218</td>
<td>1686</td>
<td>207</td>
<td>1479</td>
<td>206</td>
<td>0</td>
<td>1.000</td>
<td>0.878</td>
<td>0.122</td>
<td>0.123</td>
</tr>
</tbody>
</table>

Table F.1 Data for Various Cutoff Distances
predict accurately for any given number of articles positively classified. The nominal model follows:

\[ y = -0.0022x^2 + 1.202x - 2.4901 \]

While the DTW model can be estimated according to:

\[ y = -0.0016x^2 + 0.9474x + 4.171 \]

These models both have \( R^2 \) values in excess of 0.995 meaning they are excellent predictors. In practical terms, we can use these equations to estimate what a hypothetical customer would receive. If a customer wanted the top 200 articles from the set we are considering, \(-0.0016(200)^2 + 0.9474(200) + 4.171\), would tell us the number of correct articles, here being 129, or 64.5%. On the other hand, the Nominal model would garner us \(-0.0022(200)^2 + 1.202(200) - 2.4901 = 149\) articles, or 74.5%.

There is another story in this chart however. The light purple line at the bottom of Figure F.1 indicates how much overlap there is in the value being predicted by the nominal and DTW models. There is very little. The darker purple line indicates the accuracy level of the predictions made by ONLY the DTW model. These are articles correctly classified by the new model that would previously have not been picked up by the old one.

Figure F.2 shows how accurate the predictions that are unique to the weighted average model are. This version gets briefly as low as 0% accuracy near the highest number of predictions made, but is very highly effective at lower cutoff points.

\[
y = 9E-06x^2 - 0.0066x + 1.184
\]

\[ R^2 = 0.95562 \]

**Figure 4.5 Unique Recommendation Accuracy**

The polynomial regression done on this curve gives the equation

\[ y = 0.0000009x^2 - 0.0066x + 1.184 \]

which will show the accuracy of unique predictions at any given point.
In more direct terms, we can express the tradeoff the client makes in the form of a tradeoff between pickup rate (sensitivity) and the precision of the recommendations. The two have a correlation of -0.95, telling us that they move very consistently in opposite directions. Figure F.3 illustrates this relationship.

Regression analysis shows us the linear nature of these two factors. Change in pickup rate is expressed as:

\[ y = 0.0028x - 1.562 \]

with change in precision being expressed by:

\[ y = -0.0031x + 2.9842 \]

These are best understood in the form of an example. Consider a client who is willing to accept no less than a 65% pickup rate and does not care about the level of false positives they are presented with. By the above formula:

\[ 0.65 = 0.0028x - 1.562 \]

\[ x = 790 \]

Meaning that 790 should be taken as out cutoff value for this prediction. We can also use the accompanying formula to state that the accuracy level for those predictions will be 

\[ -0.0031(790) + 2.9842 = 0.535 \]

or 53.5% precise.

Finally, we can look at the ROC curve of the data. This plots TPR against FPR. For the DTW model at 30 minutes after first activity, the area under the ROC curve is 0.9, a very positive result. While generally a good way of examining model efficacy, the low value placed on FPR by NewsWhip’s clients mean that this is less effective than usual and another metric will be used for more general comparison.
Figure 4.7 DTW ROC Curve at 30 Minutes

Model Comparison for Various Time Intervals

Figures F.5 to F.8 show how the relationship between the DTW model and nominal count model at various times after first activity.
Figure 4.9 Model Comparison at 45 mins

Figure 4.10 Model Comparison at 60 mins

Figure 4.11 Model Comparison at 90 mins
Figure F.7 shows that only after 60 minutes does the DTW model match the performance level of the nominal model. This is too long to be of any practical use to NewsWhip.

**Alternate Models and the Impact on Accuracy**

Here, the alternative models outlined in Section 2.5 are analysed in terms of their calculation time reduction and accuracy loss. For simplicities sake, the AUC for 350 articles proposed, as in Figure F.1, will be considered the determining factor of model efficacy.

**Sampling**

By taking a random sample of articles we can control very closely how long the calculation will take, this will come at the expense of accuracy. This method simply ignores as much of the training data as we want.

Figure 3.3, page 15 shows the direct linear relationship between number of train articles considered and calculation time taken. It follows the formula:

\[ y = 1.7025x + 0.8907 \]

Examining the change in accuracy in the form of model AUC for 350 predictions shows us the price we pay for this time reduction. Samples of 10, 20, 30, 40, 50, 100, 150, 200, 250 and 300 were taken six times and the average model accuracy was measured for each.

Sporadically, the small sample tests perform better than the large ones. This behavior is inconsistent and the potential drops in accuracy exceed the potential gains. For a professional product, this level of unreliability is unacceptable. Sample sizes of around 150
do seem to reach a relative level of consistency. While this potentially halves calculation time, the accuracy level is still lower than that of the existing NewsWhip model.

**Comparison By Categorization**

Every article is categorized according to certain types of article. The behavior patterns for these categories are outlined in Appendix G. This version of the model will only find the dissimilarity an article holds for other articles with the same category tags. With 1,892 test articles and 380 train articles, there are 718,960 DTW calculations to be made. Category pairing eliminates 110,960 of these.

The performance of the model does fall significantly in this model however. As can be seen in Figure F.10, the AUC for weighted average DTW dissimilarity is 13,442, significantly lower than the nominal model, whose AUC is 16,391.

![Model Comparison Graph](image)

**Figure 4.13 Categorisation Model Comparison**

**Comparison to Average Article**

It is possible to theoretically reduce the training set to a single line. By finding the average level of activity across the entire training set we can find a single line approximation of the set.
G. The Mathematics of Virality

To attempt to quantify the value of a social media interaction we can look at the basic mathematics of virality, based on the attributes of the publisher and the article.

Publisher power (P) refers to the general strength of the publisher. This translates into the number of page views a story would be expected to receive. The inherent "shareability" of an article (S) can represent the likelihood that a reader will share this story on social media. The number of friends (F) that the reader has is the maximum number of people that can see this shared article. "Clickability" (C) attempts to estimate how tempting an article headline or main image would be to a prospective reader.

The virality of the article can be expressed as \( S \times F \times C \). For illustration purposes, we can assign hypothetical numbers to these values. Making \( S = 0.01 \), \( F = 250 \), \( C = 0.05 \), we can say that 1% of readers will share the article, the average number of friends a reader has is 250 and 5% of those people will click the article. From these numbers if \( P \) is 10,000, we should expect 100 social media interactions, that these will reach a potential 25,000 people, 1,250 of those will go on to read the article. This is a 12.5% increase in readership due to virality. Of the 1,250 second degree readers, a further 12.5% of that number should be expected to translate into readers and so on. Using this we can express total viral growth (up to \( N \) degrees of readership) as:

\[
\sum_{n=1}^{N} P \times (S \times F \times C)^n
\]

It is possible for \( SFC \) to exceed 1. This is the point at which a story grows at an exponential rate and the above formula will only cease to apply once the article has reached saturation.

In reality this is more complicated than a single number would indicate. Publisher "reputation" also holds influence over the number of click-throughs, the number of shares, and importantly, the nature of those shares. Many organizations will try to maximize their C and S values. The methods by which this is accomplished can be unsustainable. This is usually done by crafting headlines that present half stories and teasers. This is sometimes referred to as the "curiosity gap" method of attracting users.

Degrees of separation of subsequent groups of readers can stunt the realistic number of \( F \). The same piece of content will be shared by mutual friends or followers of posters and these should not be considered new opportunities for readers.

Facebook's method of presenting content also deserves mention as a disruptive factor in this model. Twitter displays content indiscriminately based purely on time of post. Facebook discriminates based on the model outlined in Section 2.4. As such, there is a chance that a friend sharing an article will not be presented with the activity. This greatly limits the F variable.

Publisher Specific Virality

We can use the publisher insights mentioned in Section 3.6 to hypothesize a more specific definition of virality for each publisher. If we use the average level of activity for a publisher as its "expected activity" this can serve as a surrogate value for S in the above equations. Any article that exceeds that value should have a final page view count significantly higher than those with lower level of social activity. Unfortunately, page views for the articles tracked are not available so we are unable to determine the exact benefit of social media activity.
## Top 10 Publishers by Twitter Activity

<table>
<thead>
<tr>
<th>Publisher</th>
<th>Article Count</th>
<th>twShares/article</th>
<th>fbShares/article</th>
<th>fbLikes/article</th>
<th>fbComms/article</th>
</tr>
</thead>
<tbody>
<tr>
<td>huffingtonpost.com</td>
<td>59</td>
<td>91.17</td>
<td>1225.88</td>
<td>3215.08</td>
<td>930.75</td>
</tr>
<tr>
<td>bbc.co.uk</td>
<td>31</td>
<td>91.08</td>
<td>39.86</td>
<td>28.73</td>
<td>20.97</td>
</tr>
<tr>
<td>g1.globo.com</td>
<td>82</td>
<td>59.32</td>
<td>13.49</td>
<td>28.00</td>
<td>9.46</td>
</tr>
<tr>
<td>abcnews.go.com</td>
<td>39</td>
<td>54.51</td>
<td>23.58</td>
<td>45.66</td>
<td>13.64</td>
</tr>
<tr>
<td>mashable.com</td>
<td>3</td>
<td>53.93</td>
<td>20.85</td>
<td>33.03</td>
<td>5.27</td>
</tr>
<tr>
<td>npr.org/</td>
<td>6</td>
<td>42.27</td>
<td>243.51</td>
<td>696.47</td>
<td>258.47</td>
</tr>
<tr>
<td>20minutos.es/</td>
<td>71</td>
<td>38.42</td>
<td>24.85</td>
<td>75.32</td>
<td>6.00</td>
</tr>
<tr>
<td>forbes.com/</td>
<td>27</td>
<td>32.41</td>
<td>5.69</td>
<td>4.63</td>
<td>1.59</td>
</tr>
<tr>
<td>youtube.com/</td>
<td>85</td>
<td>32.10</td>
<td>75.14</td>
<td>145.00</td>
<td>56.44</td>
</tr>
<tr>
<td>businessinsider.com</td>
<td>22</td>
<td>31.34</td>
<td>66.76</td>
<td>127.05</td>
<td>65.83</td>
</tr>
</tbody>
</table>

Table G.1 Top 10 Publishers by Twitter Activity

## Top 10 Articles by Facebook Shares

<table>
<thead>
<tr>
<th>Publisher</th>
<th>Article Count</th>
<th>twShares/article</th>
<th>fbShares/article</th>
<th>fbLikes/article</th>
<th>fbComms/article</th>
</tr>
</thead>
<tbody>
<tr>
<td>huffingtonpost.com</td>
<td>59</td>
<td>91.17</td>
<td>1225.88</td>
<td>3215.08</td>
<td>930.75</td>
</tr>
<tr>
<td>medpagetoday.com</td>
<td>1</td>
<td>0.17</td>
<td>344.00</td>
<td>167.17</td>
<td>90.07</td>
</tr>
<tr>
<td>ijreview.com/</td>
<td>5</td>
<td>5.92</td>
<td>267.73</td>
<td>1223.29</td>
<td>222.31</td>
</tr>
<tr>
<td>npr.org/</td>
<td>6</td>
<td>42.27</td>
<td>243.51</td>
<td>696.47</td>
<td>258.47</td>
</tr>
<tr>
<td>hindustantimes.com/</td>
<td>15</td>
<td>1.03</td>
<td>121.24</td>
<td>67.61</td>
<td>24.59</td>
</tr>
<tr>
<td>talkingpointsmemo.com/</td>
<td>10</td>
<td>5.20</td>
<td>76.19</td>
<td>61.54</td>
<td>42.39</td>
</tr>
<tr>
<td>youtube.com/</td>
<td>85</td>
<td>32.10</td>
<td>75.14</td>
<td>145.00</td>
<td>56.44</td>
</tr>
<tr>
<td>businessinsider.com/</td>
<td>22</td>
<td>31.34</td>
<td>66.76</td>
<td>127.05</td>
<td>65.83</td>
</tr>
<tr>
<td>nytimes.com/</td>
<td>4</td>
<td>25.47</td>
<td>61.54</td>
<td>118.88</td>
<td>38.44</td>
</tr>
<tr>
<td>birminghammail.co.uk/</td>
<td>3</td>
<td>1.85</td>
<td>59.24</td>
<td>72.93</td>
<td>104.41</td>
</tr>
</tbody>
</table>

Table G.2 Top 10 Articles by Facebook Shares

Tables G.1 and G.2 show top publishers by Twitter and Facebook activity respectively. Clearly, in the data considered, The Huffington Post has had the largest presence on both social media platforms. Though there is a correlation between presences on the two platforms, some publishers have a user base that is more represented on one platform. The BBC for example is ranked 2nd on Twitter but 17th on Facebook.

### Category Specific Virality

Similarly to how publisher data was analysed, we can derive certain insights about the level of engagement with different categories of content and use these to benchmark expected activity.

Table G.3 Shows the top 10 categories by Twitter activity while Table G.4 shows top categories by Facebook shares. Again we can see how the markets for different types of content are divided between the two social platforms.
<table>
<thead>
<tr>
<th>Category</th>
<th>Article Count</th>
<th>twShares/article</th>
<th>fbShares/article</th>
<th>fbLikes/article</th>
<th>fbComms/article</th>
</tr>
</thead>
<tbody>
<tr>
<td>News</td>
<td>987</td>
<td>27.18</td>
<td>69.53</td>
<td>149.55</td>
<td>58.70</td>
</tr>
<tr>
<td>Tech</td>
<td>209</td>
<td>11.67</td>
<td>6.59</td>
<td>6.88</td>
<td>4.58</td>
</tr>
<tr>
<td>Culture</td>
<td>131</td>
<td>9.77</td>
<td>62.63</td>
<td>183.27</td>
<td>44.89</td>
</tr>
<tr>
<td>Business</td>
<td>329</td>
<td>9.30</td>
<td>15.13</td>
<td>38.83</td>
<td>14.72</td>
</tr>
<tr>
<td>Gossip</td>
<td>225</td>
<td>9.28</td>
<td>56.62</td>
<td>161.60</td>
<td>41.96</td>
</tr>
<tr>
<td>Politics</td>
<td>184</td>
<td>5.66</td>
<td>27.57</td>
<td>73.94</td>
<td>20.84</td>
</tr>
<tr>
<td>The Wire</td>
<td>162</td>
<td>4.80</td>
<td>15.04</td>
<td>42.32</td>
<td>18.56</td>
</tr>
<tr>
<td>Arts</td>
<td>28</td>
<td>4.05</td>
<td>39.29</td>
<td>102.99</td>
<td>27.95</td>
</tr>
<tr>
<td>Sports</td>
<td>220</td>
<td>4.02</td>
<td>2.50</td>
<td>5.01</td>
<td>1.92</td>
</tr>
<tr>
<td>Marketing</td>
<td>44</td>
<td>3.78</td>
<td>0.96</td>
<td>0.95</td>
<td>0.12</td>
</tr>
</tbody>
</table>

Table G.3 Top 10 Categories by Twitter Activity

<table>
<thead>
<tr>
<th>Category</th>
<th>Article Count</th>
<th>twShares/article</th>
<th>fbShares/article</th>
<th>fbLikes/article</th>
<th>fbComms/article</th>
</tr>
</thead>
<tbody>
<tr>
<td>News</td>
<td>987</td>
<td>27.18</td>
<td>69.53</td>
<td>149.55</td>
<td>58.70</td>
</tr>
<tr>
<td>Culture</td>
<td>131</td>
<td>9.77</td>
<td>62.63</td>
<td>183.27</td>
<td>44.89</td>
</tr>
<tr>
<td>Gossip</td>
<td>225</td>
<td>9.28</td>
<td>56.62</td>
<td>161.60</td>
<td>41.96</td>
</tr>
<tr>
<td>Politics</td>
<td>184</td>
<td>5.66</td>
<td>27.57</td>
<td>73.94</td>
<td>20.84</td>
</tr>
<tr>
<td>Food</td>
<td>36</td>
<td>3.14</td>
<td>23.79</td>
<td>75.26</td>
<td>30.71</td>
</tr>
<tr>
<td>Health</td>
<td>50</td>
<td>1.54</td>
<td>22.24</td>
<td>11.87</td>
<td>6.13</td>
</tr>
<tr>
<td>Right</td>
<td>62</td>
<td>1.88</td>
<td>19.67</td>
<td>78.47</td>
<td>15.86</td>
</tr>
<tr>
<td>Pre-Viral</td>
<td>46</td>
<td>3.65</td>
<td>17.65</td>
<td>46.84</td>
<td>17.98</td>
</tr>
<tr>
<td>Reddit</td>
<td>26</td>
<td>3.40</td>
<td>16.80</td>
<td>45.03</td>
<td>17.19</td>
</tr>
</tbody>
</table>

Table G.4 Top 10 Categories by Facebook Shares
H. References


Couture-Beil, Alex. "rjson: JSON for R." R package version 0.2.13 (2013).


Gerald Jurasinski, Franziska Koebsch, Ulrich Hagemann and Anke Guenther, “flux: Flux rate change calculation from dynamic closed chamber measurements.” R package version 0.2.2, (2013)


Myers, Cory, Lawrence Rabiner, and Aaron E. Rosenberg. "Performance tradeoffs in dynamic time warping algorithms for isolated word recognition."Acoustics, Speech and


