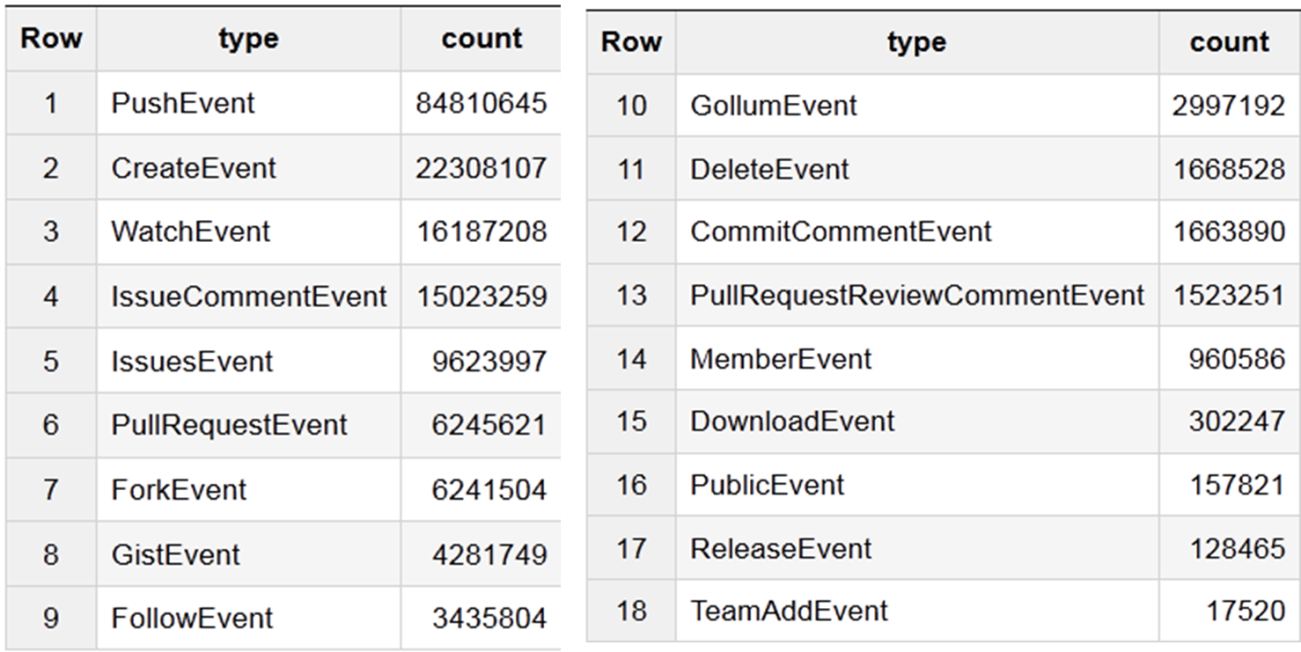
This document provides an overview of some of the data collection and analysis work conducted for the project.

**Data sources**

1. **Github Timeline on Google BigQuery**: This is a data-set which records certain events on github.com (one row per event) and is continuously updated. It began in March 2012, and as of March 2014 there were 174 million events. The data is in the form of a wide table with 199 columns, many of these columns only relate to specific types of event (e.g. a pull request has many recorded variables). BigQuery is a cloud computing service, and the fact that this data-set already exists on BigQuery facilitated us working with data at a scale which would otherwise have been more difficult – albeit at a cost charged for every query which was executed. BigQuery takes structured queries, but lacks certain features common to most SQL databases (the capacity to update a table or add a new column). This necessitated a particular approach to data analysis, involving the use of JOIN and GROUP EACH BY queries to iteratively build up tables which captured the variables of interest. Many of the queries written for BigQuery are included in bq\_queries.txt, the research process often involved exporting the results of queries from BigQuery to a local MySQL server or directly into the R statistical environment for further analysis. There are several limitations of the timeline data-set on BigQuery: it contains no information about activity prior to March 2012, it contains no information about events like commits which occur at the level of the git repository, these only appear on the timeline when commits are “pushed” to the repository, a single PushEvent can subsume many commits (and the identity of the committers).

**Figure 1**. Event types on the github timeline

1. **Github API:** Limitations with the BigQuery data-set meant that it was also necessary to collect data through Github’s API. This was achieved with Python scripts, many of which are scattered throughout the data\_analysis folder in relevant sub-folders (e.g. git\_history, github\_data\_infrastructures). Collecting data through the Github API involves specifying in much more detail the specific data which is being requested, there is also a limit to the number of requests which can be made in a given period of time. In practice, many aspects of this research involved stitching together data collected through the Github API with data aggregated from the Timeline on BigQuery.
2. **Stackoverflow:** Stackoverflow regularly releases public data dumps which describe activity on the site. One of these dumps was imported to BigQuery so that it could be cross-referenced with the data from github to ask questions like whether there is a relationship between the levels of activity around a repository on github and the number of questions being asked and answered on Stackoverflow.

**Lines of Enquiry**

**Highly skewed distributions**

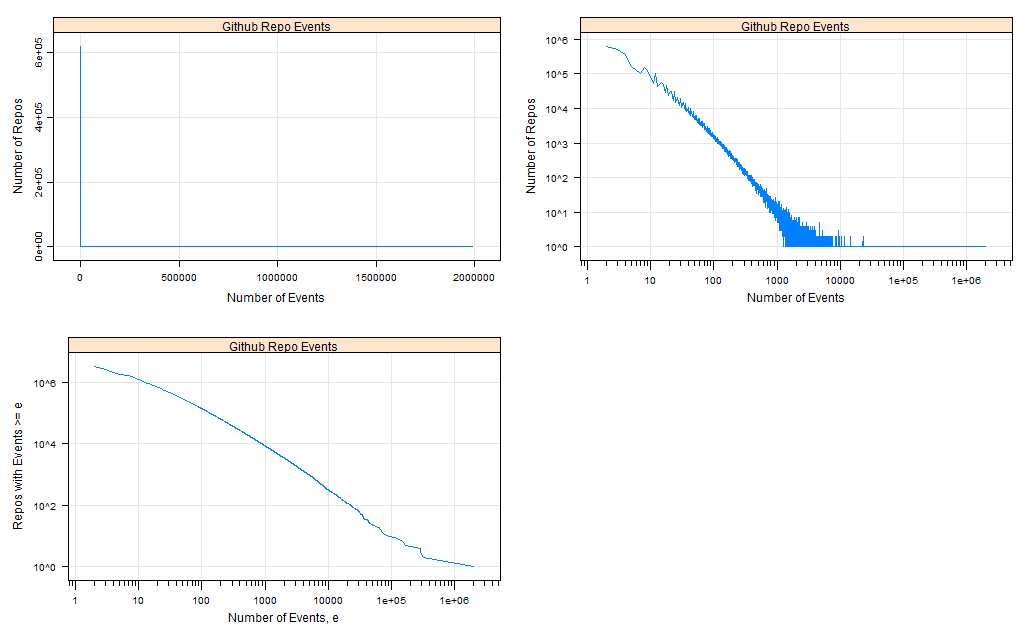
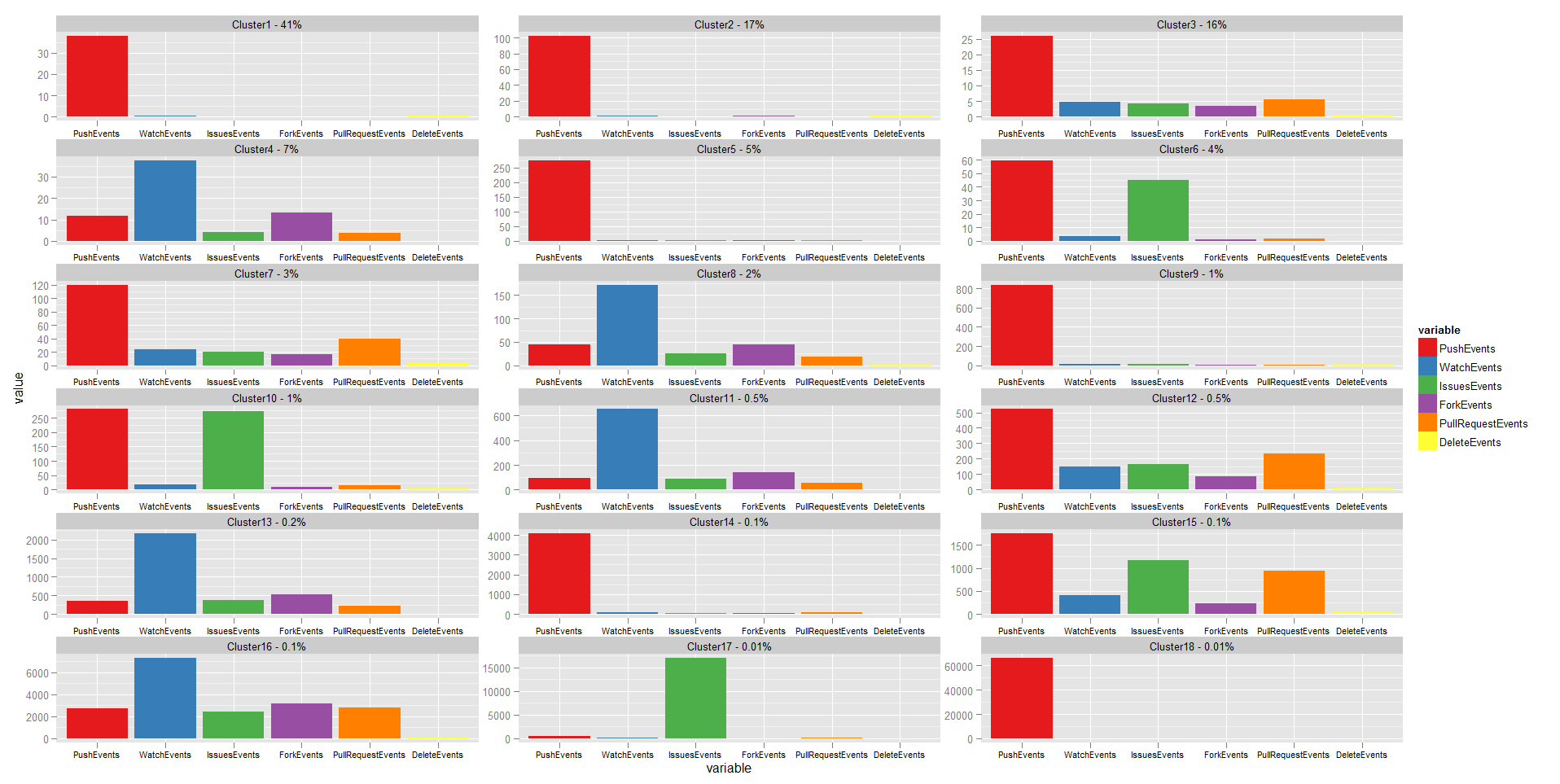
One of the first steps in the analysis was to confirm the expected prevalence of power-law type distributions of activity between repositories. This distribution sets the stage for all subsequent analyses. Of the 4 million individual repositories evident in the timeline data, around 20% had just a single event associated with them. At the other end of the scale are repositories with more than one million events. One oft-cited corollary of the power law is the 80/20 rule – for this data the most active 13% of repositories have between them 80% of all Events. The most active 0.5% of repositories account for 37.5% of all Events. As with any highly-skewed distribution – the activity ‘on github’ is happening largely on a relatively small set of repositories.

Figure 2. Distribution of events between repositories on the github timeline.

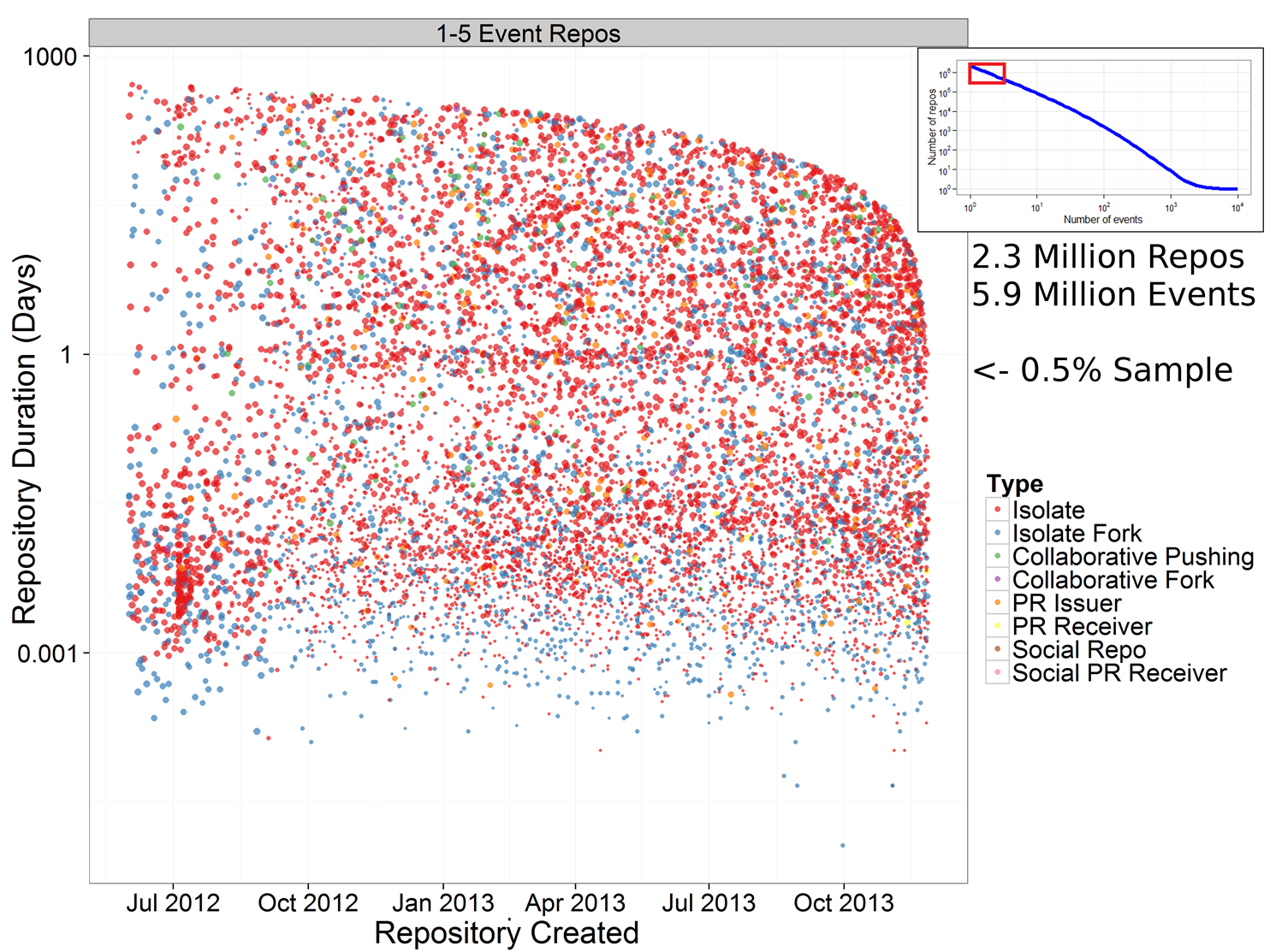
**A typology of repositories**

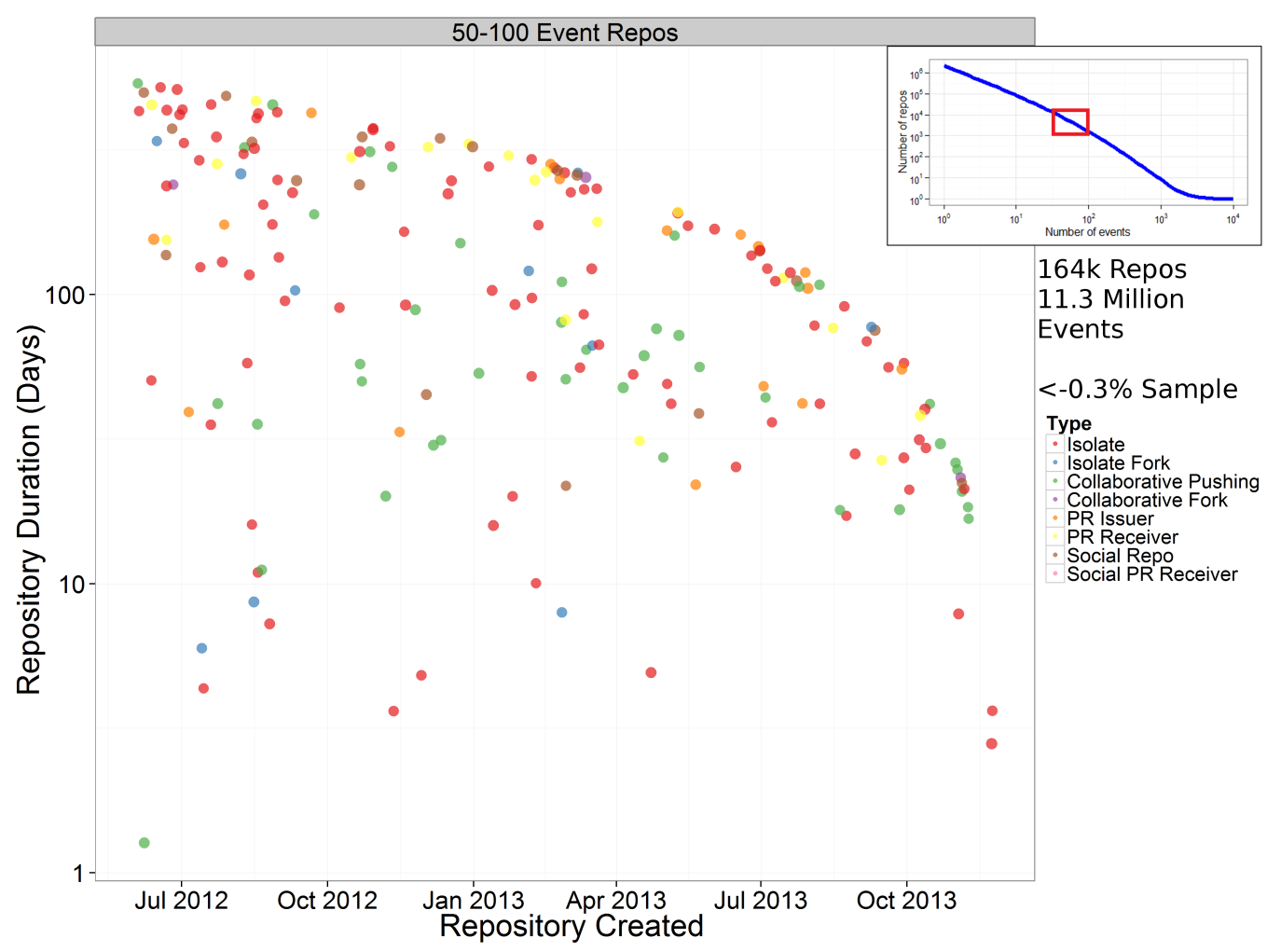
To try and make sense of the large number of repositories on github, several attempts were made to categorise these based on the number and type of events associated with them. In the early stages, [we observed a relationship between the frequency of different types of events](http://metacommunitiesofcode.org/2013/08/16/github-repo-events-census/), to the extent that many of these could be conceptualised as either “owner events” (requiring contributor access, e.g. push, create, delete) or “social events” (e.g. Watch, Fork, Issue, Pull Request) – with the frequencies for these events being correlated within these categories but not between them to the same degree. Latent Class Analysis was used to automatically cluster repositories (with more than 33 events each) according to the number of certain event types.

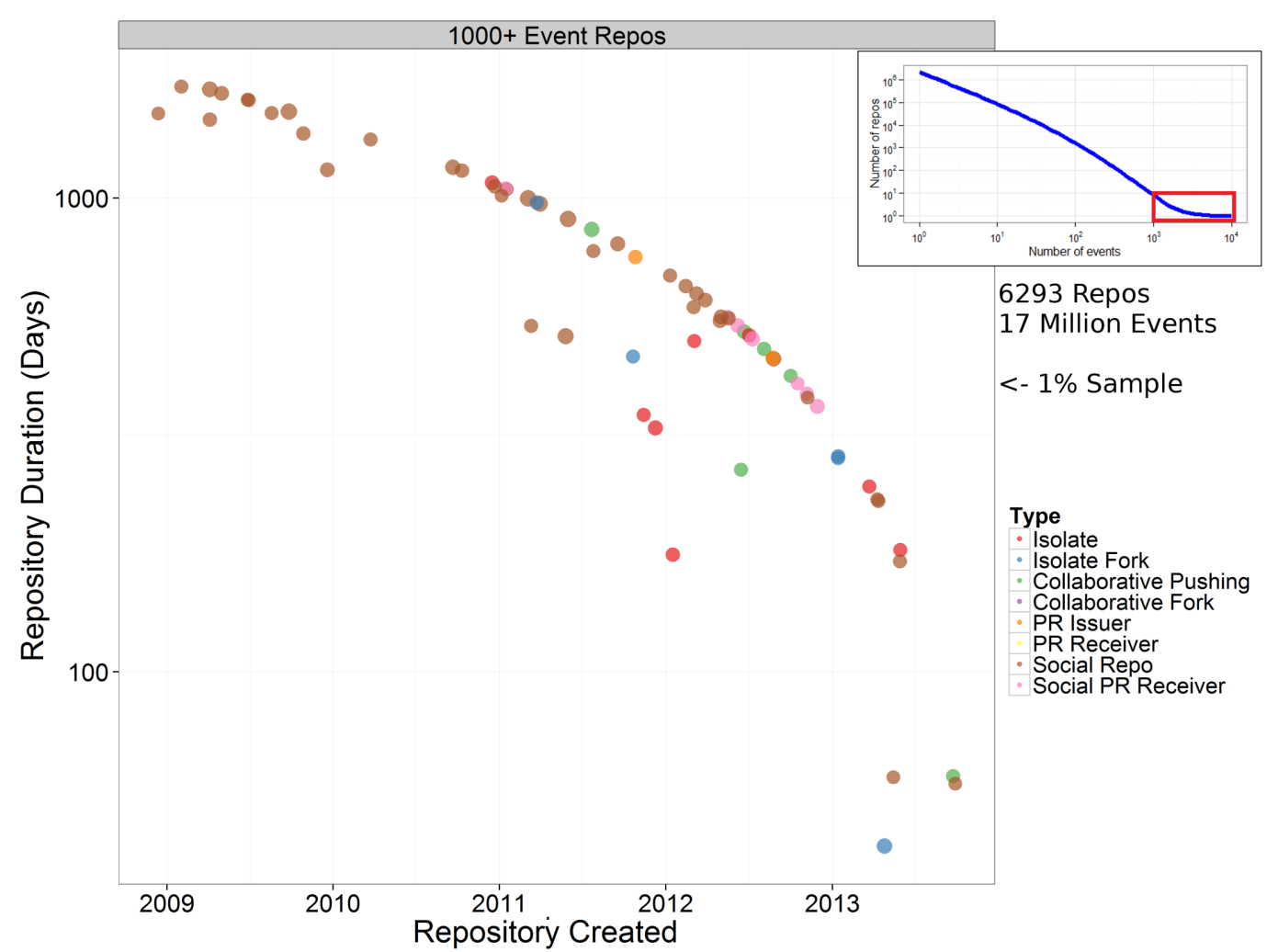


**Figure 3. 18 latent classes fitted on event counts**

Later in the project, with a better idea of the roles and relative importance of events, we categorised repositories manually to explore and visualise the relationships between their type, level of activity, and activity lifespan.





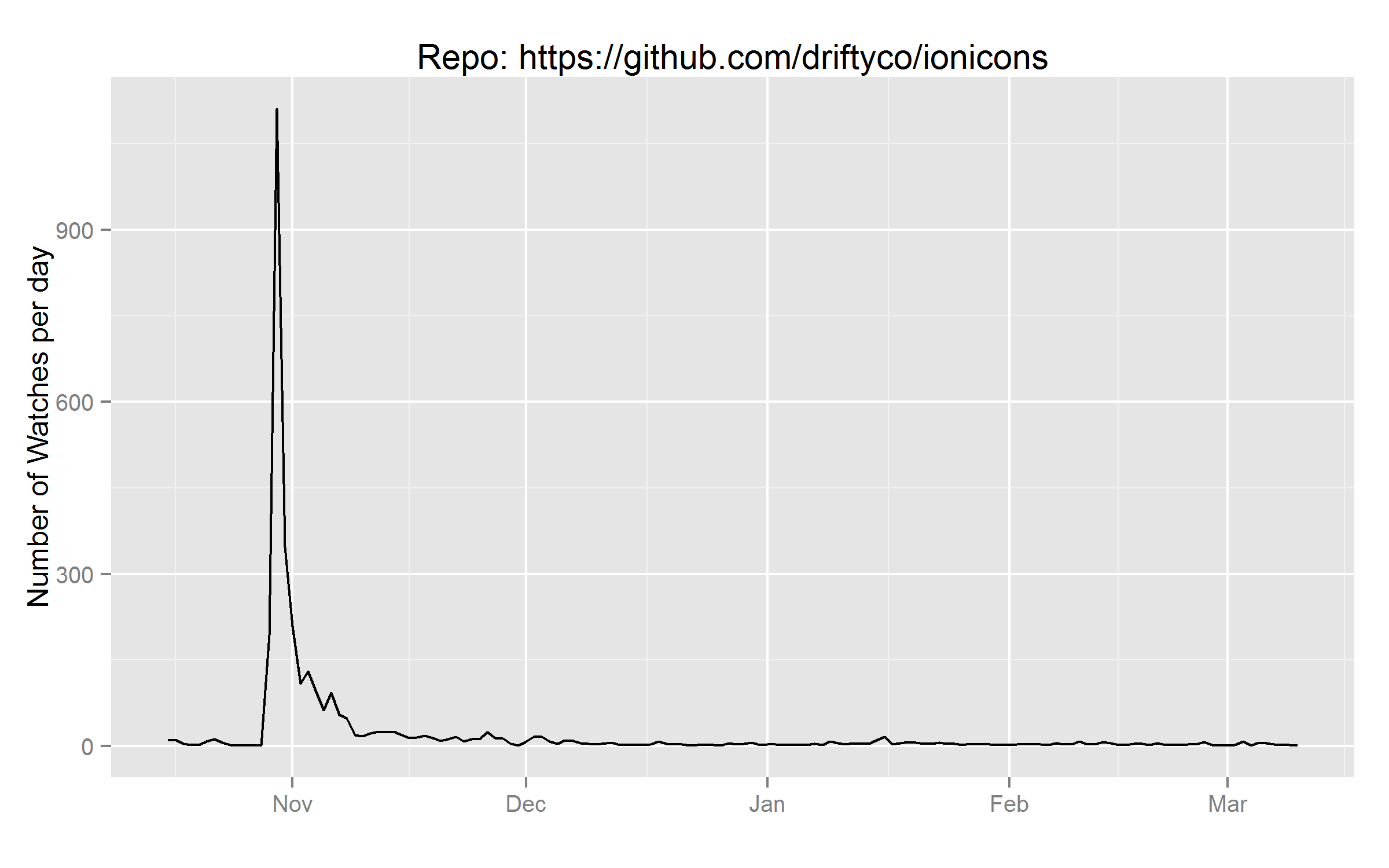


**Figure 3. Visualisation showing repository lifespan, number of events (point size) and type of repository (point colour)**

Much of the work along this line of enquiry is contained in the github\_repository\_types folder.

**Repositories over time – important events**

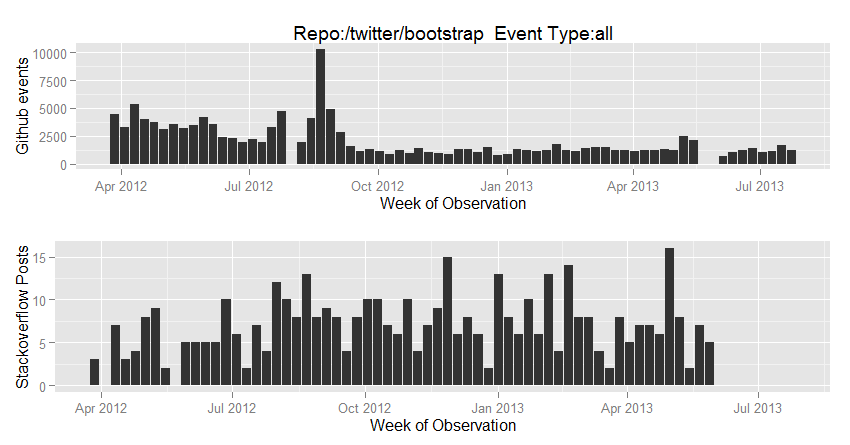
By looking at a repository’s social activity over time, it is possible to detect when important events occurred which raised its profile. It proved much more difficult to determine what these important events were. Work related to this is in the github\_social\_practices folder.



**Figure 4. Number of new “watchers” per day for the driftyco/ionicons reposository**

**Linking to Stackoverflow**

One of the ways in which we linked Github to Stackoverflow was by looking at events over time on each platform which related to a specific repository.

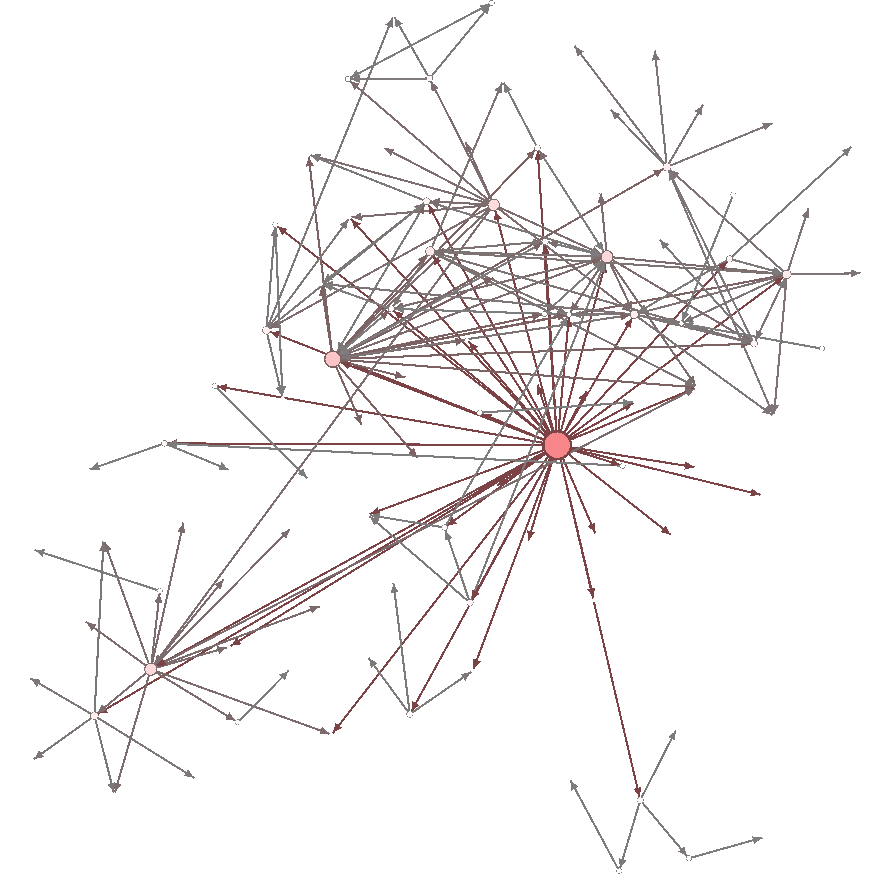
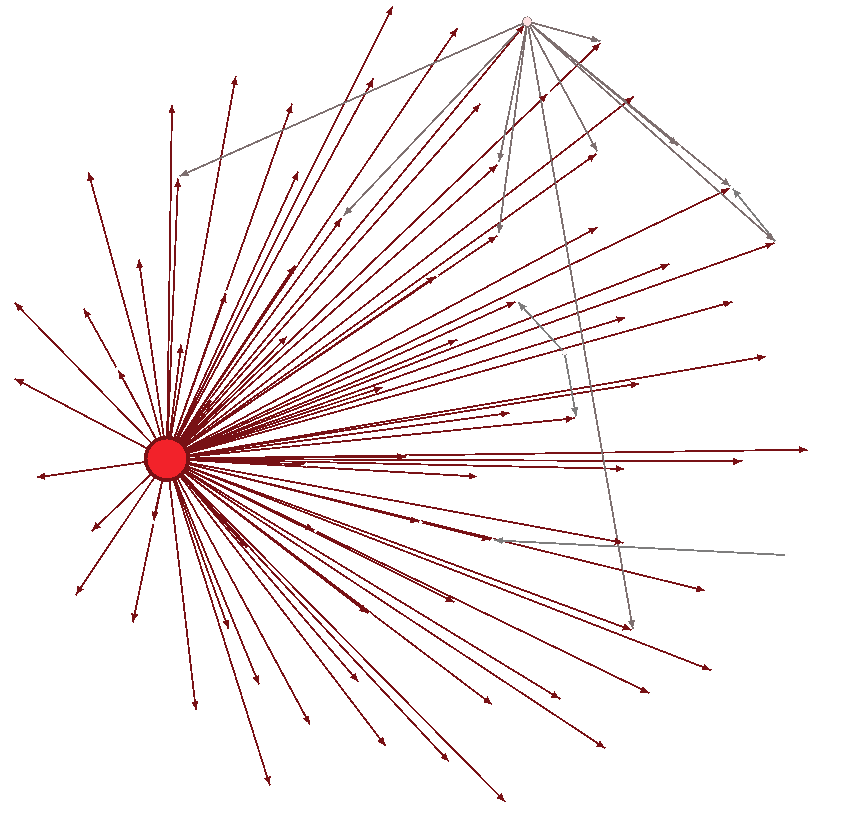


**Figure 5. Github events and Stackoverflow posts per week for the twitter/boostrap repository**

**Pull Requests**

Pull Requests are one of Github’s key innovations, they allow someone who is not a contributor to a project to contribute to its development. Any user can fork a project, make changes, and then issue a pull request – which is effectively an invitation to the parent repository to incorporate changes made by this external user. Pull requests, and the relations between forks and their parents, are the subject of a number of the files in the data\_reports folder and also posts on the [metacommunities blog](http://metacommunitiesofcode.org/).

Through analysis of pull requests on the github timeline we discovered that some projects use this mechanism internally as a form of code review (i.e. an individual who could push their changes directly instead submits them in the form of a pull request). We also considered whether pull requests could be a novel recruitment tool through which an individual could demonstrate their utility to a project un-invited. We identified around 14,000 instances where a user’s pattern of activity related to a repository was consistent with “pull-requesting their way into a project”(i.e. forking it, making changes, pull-requesting those changes back to the project, and subsequently becoming a direct contributor to the project).



**Figure 6. Two pull request networks.**

The “networks” between repositories which are formed by pull requests were also considered. As expected, these predominantly involve a single central repository which acts as the authoritative version of the project, individuals fork this repository and pull request their changes back to it. However we also noticed some instances where this was not entirely the case, figure 6 (left) shows a network with a secondary “hub” (perhaps indicating a schism in a project) and figure 6 (right) shows an altogether more distributed pull-request network.

**Organisations**

While most of our analyses were conducted at the level of the repository, we also explored github at the level of the organisation. Many projects are composed of multiple repositories, all owned by the same individual or organisation.

**What are these entities all about?**

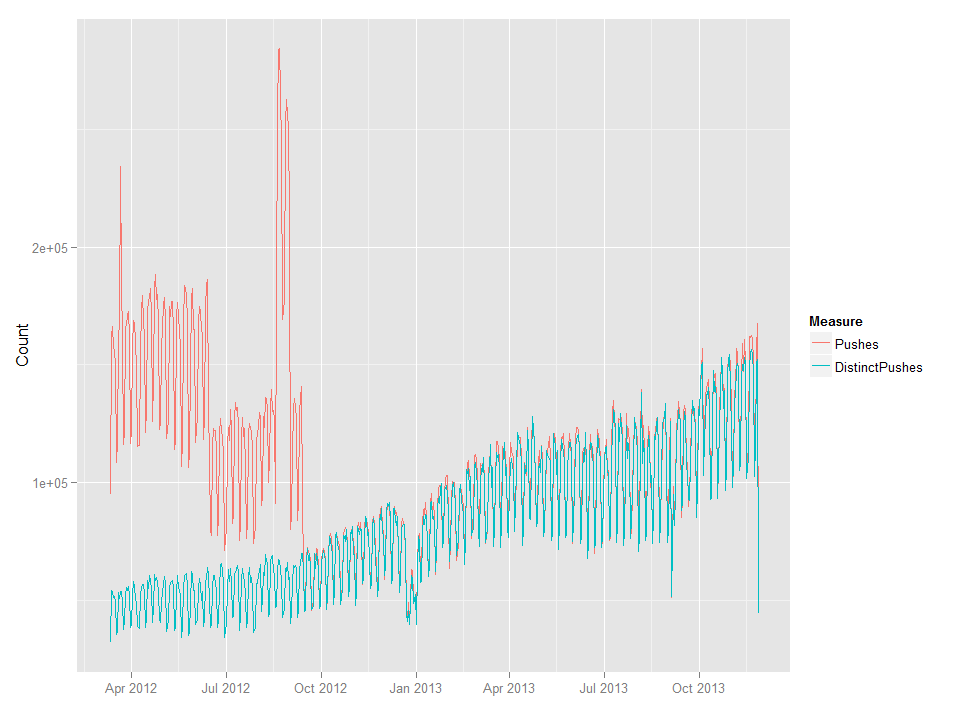
One of the questions which proved most difficult to answer was about how we could automatically determine what a repository or organisation’s purpose was. Several approaches were deployed here, including topic classification with Latent Dirichlet Allocation and machine learning on textual material associated with a repository (e.g. its readme file). The technical nature of these documents, and the fact that different projects use them in different ways (they contain licenses, installation instructions, pull request protocols) meant that it was not possible to automatically classify these entities with regard to their purpose or domain.

Some of the work on this is included in the github\_organisations/classify\_orgs and github\_repository\_topics folders.

**Data Quality**

Data on the github timeline is produced automatically and represents actions which have occurred on the platform – it is easy to assume that this data is accurate and valid. On one occasion, we found that this was not the case – with individual events being recorded multiple times in the timeline data. This issue seems to have been rife in the first 8 months of timeline data, after which it appears to have been addressed. This issue was not documented anywhere to our knowledge, and its discovery necessitated re-visiting some of our earlier analyses. This kind of ‘big data’ set cannot always be taken at face value.

We also noticed many anomalous patterns in the data which we suspect are the result of individuals learning the github system, playing with it, or testing its limitations with bots.



**Figure 7. Number of timeline PushEvents and number of genuinely distinct PushEvents per day**