TrackMeNot-so-good-after-all

Rami Al-Rfou'
ralrfou@cs.stonybrook.edu

William Jannen
wjannen@cs.stonybrook.edu

Nikhil Patwardhan
npatwardhan@cs.stonybrook.edu

December 13, 2010

Abstract

TrackMeNot is a Firefox plugin with laudable intentions - protecting your privacy. By issuing a customizable stream of random search queries on its users' behalf, TrackMeNot surmises that enough "search noise" will prevent the users' true query profiles from being discerned. However, we find that clustering queries by semantic relatedness allows us to disentangle a nontrivial subset of user queries from TrackMeNot issued noise.

1 Background

The decreasing cost of persistant media for storage, coupled with the steady rise in E-commerce, social networking, and various other online services, has led to a dramatic increase in the volume of readily available, personally identifiable information. One often overlooked source of such information is the logging of user queries by search engines.

There have been several high-profile examples of search query data being used in inappropriate ways, most notably an incident involving AOL in 2006 [4]. The company disclosed a data set comprised of information from 658,000 subscribers' search histories; it was released as a flat file into the public domain. Upon admitting their error, they removed the link, but the data was already mirrored elsewhere. This incident highlights just how personally revealing search engine queries can be, as at least one user was identified using only the content of her searches [**?**].

The average internet user may browse under the false impression that deleting cookies or diverting traffic through a proxy will protect their anonymity on the web. Neither of these measures provide complete protection. If a user wishes to utilize certain services such as Google Mail, they must log in to their personal profile. Once logged in, their queries are then tied to their accout.

TrackMeNot is a Firefox plugin that aims to protect the privacy of its users by issuing random queries on their behalf. These random queries are pulled from a variety of sources, with the intuition that providing enough "noise" around a user's true search patterns will make it impossible to disentangle the queries made by TrackMeNot from those actually made by human searchers.

2 TrackMeNot details

TrackMeNot operates completely on the client side as a Firefox plugin. Upon startup, an initial *seed list* is populated from RSS feeds and known public "popular search" lists. During execution, queries are pulled from this list and issued to popular search engines on the user's behalf.

The user is offered many parameters, which can be tuned to customize TrackMeNot behavior. For instance, the user can select which, if any, popular search engines he wishes TrackMeNot to query on his behalf. If the user only performs searches at `http://www.Google.com`, he can specify that TrackMeNot query only that site as well. The user can also specify the frequency of TrackMeNot

queries (the default average query rate is 10 queries per hour), as well as whether or not to enable *query bursts*. Query bursts are triggered by actual user queries; when TrackMeNot detects a genuine query, it responds by issuing a batch of simulated searches. In a subset of query bursts, a longer query is selected and permuted to form a set of potentially similar searches.

The basis of the TrackMeNot model is its *dynamic query list*. Starting from the initial seed list, queries are randomly marked for replacement over time. When a marked query is sent by TrackMeNot as a search engine request, the HTTP responses are parsed to identify any suitable "query-like" terms for replacement. If an acceptable term is found, it is then substituted into the dynamic list in exchange for the marked query. In this way, the list of searches will evolve over time.

TrackMeNot also attempts to simulate user browsing patterns with "selective click-through"; upon issuing a random query, it will parse the results page, using regular expression matching to identify and remove revenue generating ads. It then selects one or more of the remaining links on the page and simulates a user click.

Although TrackMeNot simulates temporal search patterns with techniques like query bursts and selective click-through, it may not sumulate contextual search patterns - sequential TrackMeNot queries are most likely semantically unrelated. Individual queries are selected at random from a 100-200 term dynamic query list. So while it is likely that the dynamic query list contains related searches, as long as searches are selected individually at random, there is no guarantee that related terms will be issued in temporal proximity. This is the aspect of the TrackMeNot design we wish to exploit.

3 Design overview

Our overall design can be logically divided into three parts. The first part deals with logging all user Google Search queries as well as those fired from TMN. The second part comprises the computation of a similarity matrix on this set of queries using a suitable semantic distance measure. The last part consists of clustering this data using a suitable technique to get a visual representation of the data from which conclusions can be deduced.

4 Data Gathering

To gather the data we ran TMN with the default configuration, except we enabled the logging feature and disabled the other search engines, focusing on google only. Moreover, we disabled the instant search feature as it produces incomplete queries. However, this does not give google a chance to distinguish TMN queries as they appear as inserted by the keyboard. On the google side, we made a new google account, **cse509tmn**, and enabled the web history for verification.

To collect the user queries we intercepted a proxy server that log all the http requests made by the browser. By the help of TMN log we were able to distinguish between the user queries and the ones generated by TMN.

The plugin was running for at least 3 days on three machines with three different users.

5 Semantic Distance Measure

The basic idea here was to find how *similar* each search query was to every other query, making no distinction based on its origin (user or TMN). To do this, we needed a *semantic* distance measure. We did not implement our own measure, but instead explored the available choices from other research groups. After examining the feasibility of using different available measures, we settled on two of them. One of them is DISCO[1], which is a Java package, and the other is Google Normalized Distance. For descriptions, see sections 5.1 and 5.2 respectively. In each of these cases, we disabled the auto-complete feature of Google so that our search queries were always the exact phrases that the user or TMN requested from Google Search and never subsets of the actual queries.

[1] http://www.linguatools.de/disco/disco_en.html

5.1 DISCO

The DISCO (extracting DIStributionally related words using CO-occurrences) API allows to compute the semantic similarity between any two words by looking up those words in a pre-defined repository. For our analysis, we downloaded the Wikipedia repository available on the DISCO website to compute the distances. By looking up this local repository, the DISCO API returns a value between 0 and 1 for any pair of words that it looks up in the repository, such that the higher the value returned, the more similar the words are. In this sense, the API works like a similarity measure. We were however faced with one issue: Google search queries are typically phrases, and not single words. To overcome this, given two queries $Q1$ and $Q2$, we compared each word in $Q1$ with every word in $Q2$ and in each case chose the highest retured value. We aggregated this score over all words in $Q1$ and normalized the addition by dividing the aggregate score by the number of words in $Q1$.

5.2 Google Normalized Distance

Google Normalized Distance (GND) is a semantic similarity based on statistics. GND has a large set of vocabulary. The similarity can be calculated according to the following equation:

$$GND(Q1,Q2) = \frac{Max(\log f(Q1),\log f(Q2)) - \log f(Q1,Q2))}{\log N - Min(\log f(Q1),\log f(Q2))} \quad (1)$$

where $f(Q)$ is the number of results found by Google for query Q, and $f(Q1,Q2)$ denotes the number of results found by google for the combined query of $Q1$ and $Q2$. The more the two queries appear together the smaller the distance between them. GND is not an accurate function to measure the distance[1]. Moreover, GND is not a metric nor it satisfies the triangle inequality, and $GND(x,y)$ is not larger than 0 for every $x \neq y$.

6 Disentangling user queries

Once we have created a matrix of semantic similarities, we are tasked with deciding which queries were actually made by the user. We first perform cluster analysis to identify groups of related queries. We then use the size and distribution of the cluster assignments to label a set of user queries.

6.1 Clustering

For clustering we use the partitioning around medoids algorithm (PAM). It is conceptually similar to the well known k-means algorithm; however, rather than minimize squared Euclidean distances, it aims to minimize dissimilarity [3].

An ideal assignment of n objects into clusters would be an assignment that minimizes cluster widths while maximizing cluster separations. In other words, each element i belonging to a cluster A should be closely related the other members of cluster A. At the same time, each element i in cluster A should be unrelated to the members of all other clusters $C : C \neq A$; individual clusters should be well separated from other clusters.

Formally, we let $a(i)$ denote the average dissimilarity between an element i and all elements in i's cluster Aj. Let $d(i,C)$ denote the average similarity between an element i and all ements in a cluster C to which i does not belong: $C \neq A$. If we let $b(i)$ be the minimum $d(i,C) : C \neq A$, then

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

is a good indicator to the quality of element i's cluster assignment. If $s(i)$ is close to one, then i is well assigned. If $s(i)$ is close to negative one, then i is poorly assigned. $s(i)$ is known as the silhouette width. Choosing a number of clusters, k, that maximize the average silhouette width is the heuristic used to determine the cluster assignments [3].

For our implementation of the PAM algorithm, we used the cluster package from the R statistical programming language[5].

6.2 Classification

Our classification strategy is a simple one. It is not our goal to show that every user query can be discerned from those queries issued by TrackMeNot. What we want to show is that some nonempty subset of user queries can be identified as "real" with

high probability. For this task, we simply choose the single largest cluster or set of clusters, and label all constituent members as user-generated queries.

6.3 Validation

We have compiled and performed testing on four data sets, containing both user and TrackMeNot generated searches. The data sets are summarized in the following table.

Data Set	# Queries	Similarity Measure
test.m	79	Disco BNC
bill.m	131	Disco Wikipedia
rami.m	305	Disco Wikipedia
nikhil.m	600	Disco Wikipedia

As you can see, our data sets vary in number of queries. This was intentional, as it allows us to test the effect of window size. Data in the larger sets spans multiple sessions.

To validate our classifier, we first computed a similarity matrix using the Disco Wikipedia distance measure. From this matrix, we performed clustering using PAM, as decribed in Section 6.1. Our classifier then identifed the largest cluster or set of clusters, assigning the label 'U' to these queries. We finally compare the assigned label against the real value to calculate the precision and recall over our validation set.

7 Results

Our results are very promising. We are able to successfully identify a subest of the user generated queries with a high precision for small window sizes. It is important to note that we make no attempt to identify all user queries. We only wish to identify the . Thus, our recall is very low, even for small window sizes.

Data Set	Precision	Recall
test.m	1.0	0.147
bill.m	1.0	0.25
rami.m	0.867	0.188
nikhil.m	0.125	0.067

8 Conclusions

For small windows, it works.

for large windows, trackmenot clusters appear because with enough samples, they will have largeer coverage of their query list

google normalized distances is not applicable to furll queries, as used. it must be applied per keyword

9 Future work

By operating as a Firefox plugin, TrackMeNot does not protect against search engines tracking time of use. As a standalone application, TrackMeNot could be run in the background, even while a user is not actively browsing. This would not only provide the same semantic search noise as the current TrackMeNot implementation, but it would serve to protect the temporal search habits of users.

TrackMeNot does not differ across machines.

Over long periods of time, TrackMeNot makes related queries.

We take time of query into account. cluster different windows.

References

[1] R. Cilibrasi and P. M. B. Vitányi. The google similarity distance. *CoRR*, abs/cs/0412098, 2004.

[2] D. C. Howe and H. Nissenbaum. Trackmenot: resisting surveillance in web search. Technical report, NYU, 2008.

[3] L. Kaufmann and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons Ltd., Chinchester, New York, Weinheim, 1990.

[4] D. Kawamoto and E. Mills. Aol apologizes for release of user search data. CNET, August 2006.

[5] M. Maechler, P. Rousseeuw, A. Struyf, and M. Hubert. Cluster analysis basics and extensions. Rousseeuw et al provided the S original which has been ported to R by Kurt Hornik and has since been enhanced by Martin Maechler: speed improvements, silhouette() functionality, bug fixes, etc. See the 'Changelog' file (in the package source), 2005.

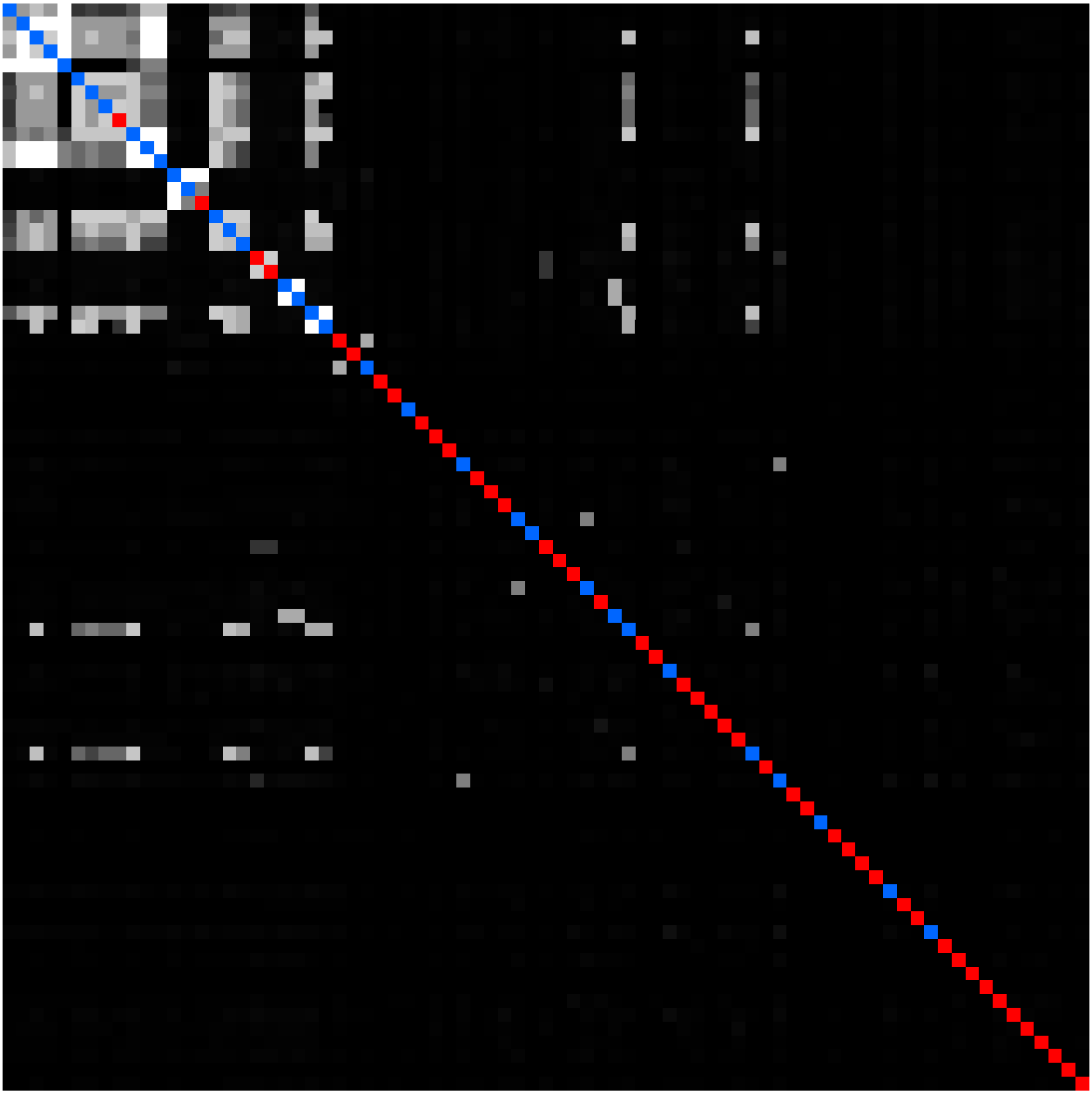

Figure 1: A pictorial representation of the `test.mat` clustering. Position (i, j) represents the similarity between query i and query j - the lighter the shade, the more similar the queries are. Each query's class is displayed along the diagonal. If (i, i) is red, then query i was generated by TrackMeNot. If (i, i) is blue, then the query is genuine. We see that many queries are semantically unrelated. The largest cluster, seen in the top left of the graph, has 5 user queries and 0 TrackMeNot queries.

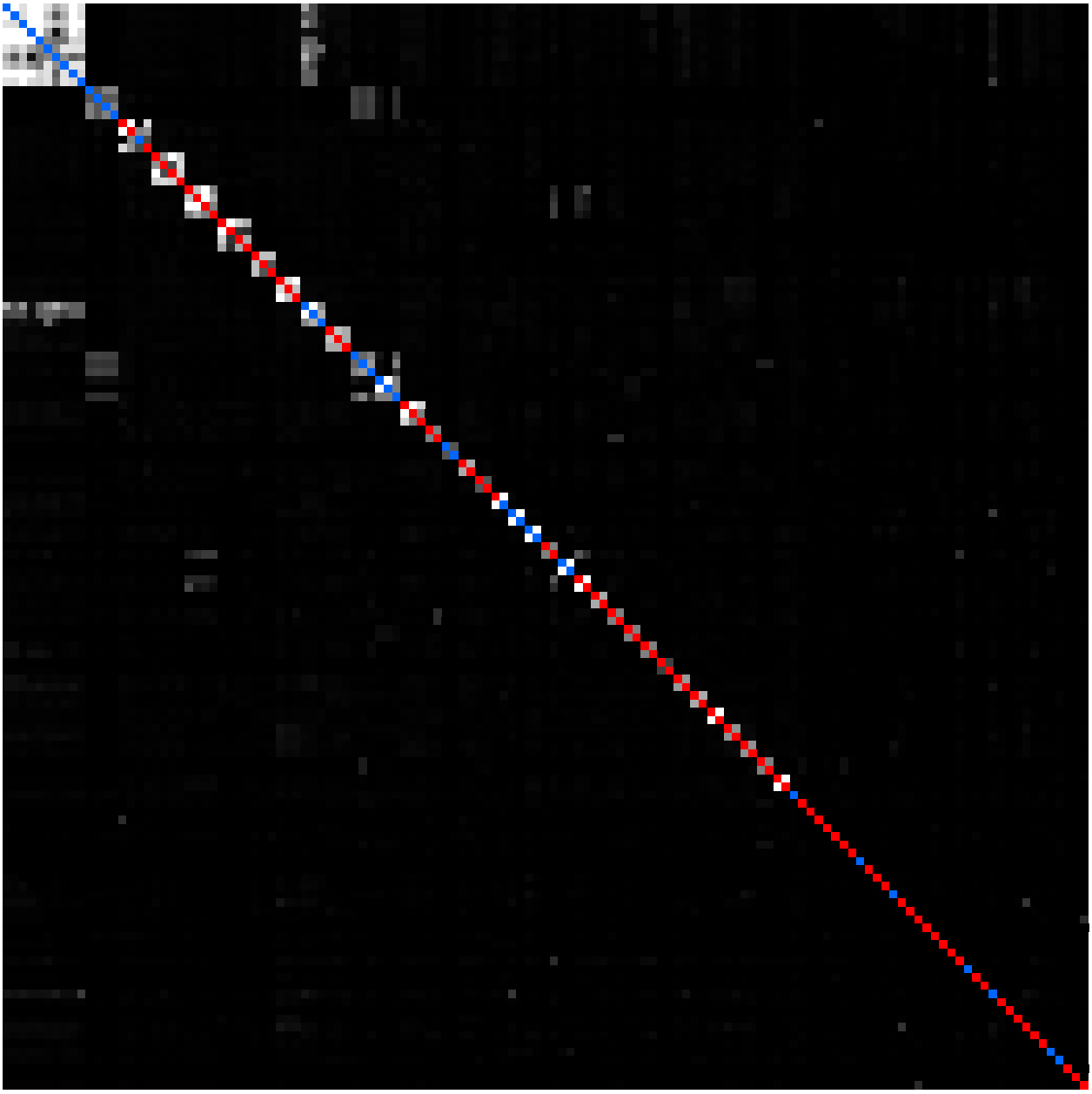

Figure 2: A pictorial representation of the `bill.mat` clustering. Position (i, j) represents the similarity between query i and query j - the lighter the shade, the more similar the queries are. Each query's class is displayed along the diagonal. If (i, i) is red, then query i was generated by TrackMeNot. If (i, i) is blue, then the query is genuine. We see that many queries are semantically unrelated. The largest cluster, seen in the top left of the graph, has 10 user queries and 0 TrackMeNot queries.

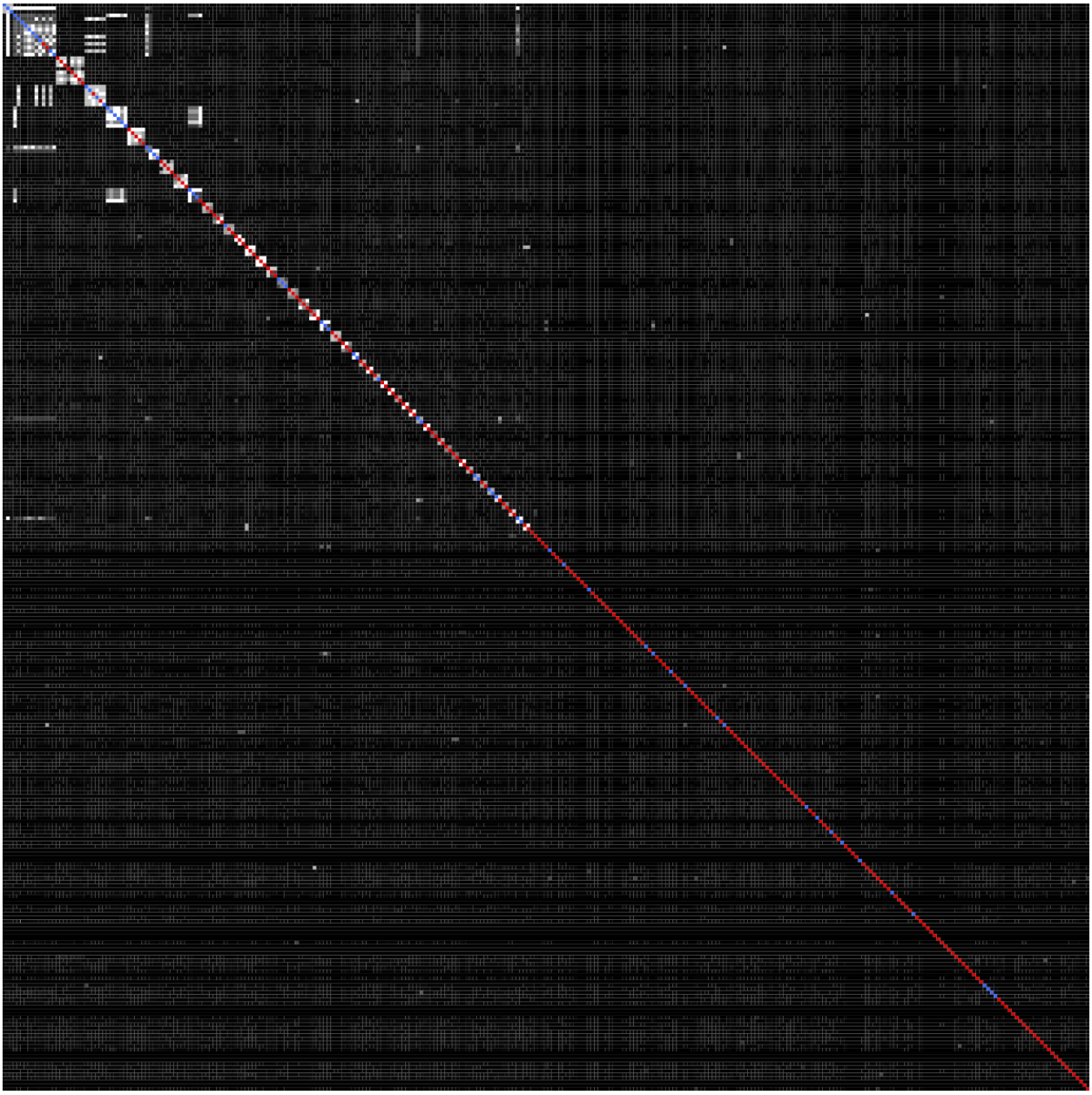

Figure 3: A pictorial representation of the `rami.mat` clustering. Position (i, j) represents the similarity between query i and query j - the lighter the shade, the more similar the queries are. Each query's class is displayed along the diagonal. If (i, i) is red, then query i was generated by TrackMeNot. If (i, i) is blue, then the query is genuine. We see that many queries are semantically unrelated. The largest cluster, seen in the top left of the graph, has 13 user queries and 2 TrackMeNot queries.

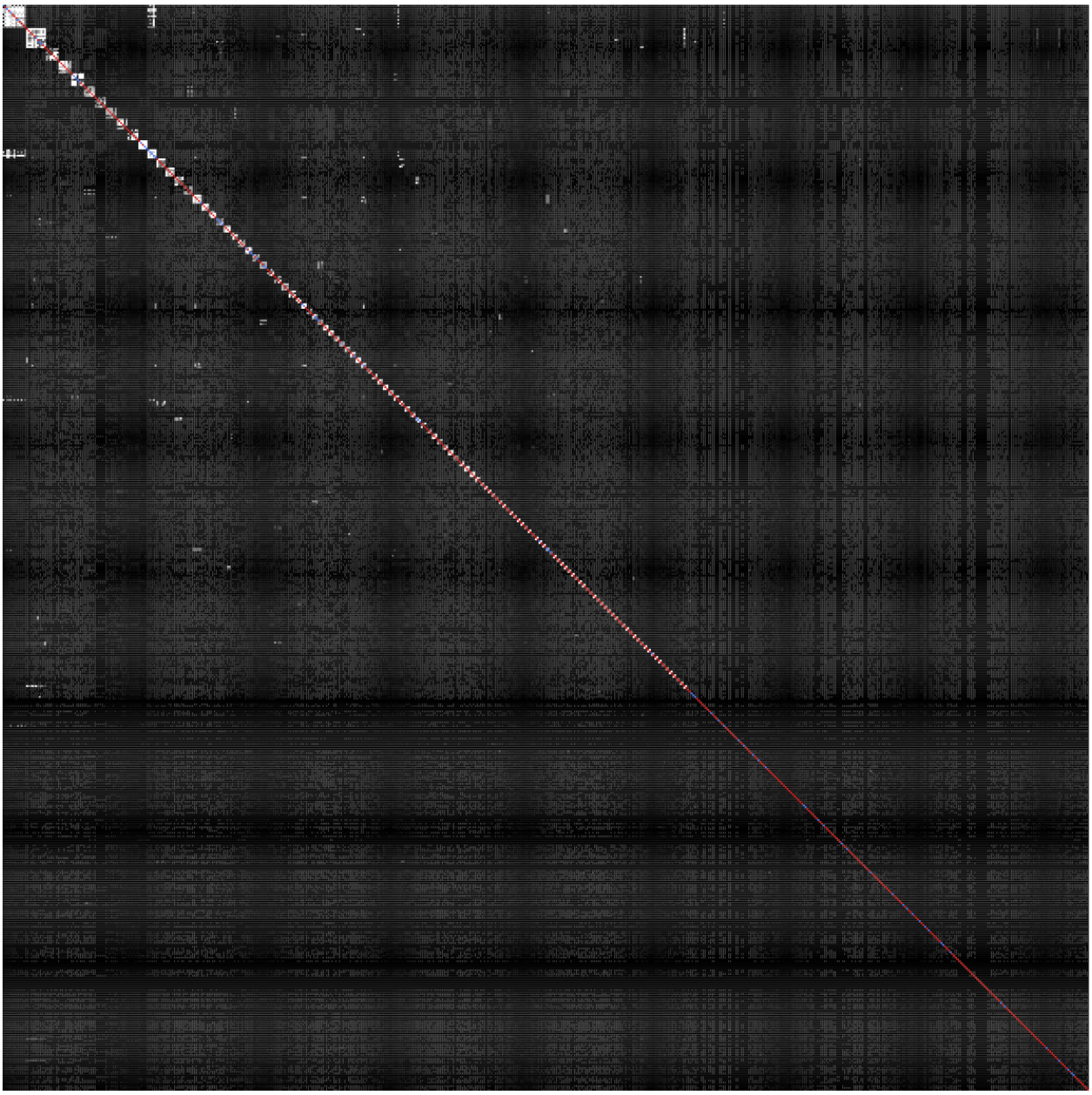

Figure 4: A pictorial representation of the `nikhil.mat` clustering. Position (i, j) represents the similarity between query i and query j - the lighter the shade, the more similar the queries are. Each query's class is displayed along the diagonal. If (i, i) is red, then query i was generated by TrackMeNot. If (i, i) is blue, then the query is genuine. We see that many queries are semantically unrelated. The largest cluster, seen in the top left of the graph, has 4 user queries and 9 TrackMeNot queries.