ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Travel: Shared topics among locations

Automatically detect topics in the city and the associated locations

Optional Semesterproject by Renato Kempter

Supervised by Dr. Pearl Pu
Laboratory of Human Computer Interaction

08.01.2015

Contents

1 Abstract

We often have a subject, a taste, a style or an interest that guides us to locations we like. Using geo-localized photos from instagram and their associated hash-tags, we propose a batch processing system that: a) Automatically clusters the a set of instagrams based on the geographical coordinates, b) Extracts the most trending topics and c) Finally generates sets of locations that share the same topic. Through a web application, users can then browse the different topics and their corresponding locations.

2 Introduction

Instagram[1] is a mobile application used to upload photos and share them with friends and followers. Users have the ability to attach hash-tags to the posted image, that serve as topics and cluster similarly tagged photos together. Optionally, photos can be geo-localized, either using a location name or coordinates from the integrated GPS of the device. Instagram is a highly popular mobile application with over 300 million users. Instagrams can be published either privately or to the entire world. For the second choice, instagram provides an API that allows access to all publicly posted instagrams[2].

Due to its large user base, instagrams are published all over the world and the application is heavily used while traveling.[3]. Because of the quantity of data accessible through the Application Programming Interface (API) and the recent success of startups (e.g. JetPac[4], acquired by Google) using the user generated content to aggregate additional information about locations, we were motivated to extract sets of location that share a common topic. A topic is similar to a description using a bag of words. An example for a topic we extracted from our data set is "creative, diy, vintage". Using topics for travel recommendation has multiple advantages:

1. Topics can be dynamic: Compared to both tags and categories, which characterize a location, topics can change over time and are an aggregation of locations. From our data set, we were able to extract topics such as "Christmas" or "Automn, fall".

2. A list of locations having something in common is an approach that has been used widely in different other areas, such as in music (Spotify Playlists) or fashion[5]. In the area of traveling, different startups and established companies are trying to motivate people to create such lists (e.g. Airbnb[6]).

Users publishing a photo with a location attached often add hash-tags to describe the taken picture, the location or the mood. Our work is based on the assumption that the location where the picture was taken and the description with hash-tags are related. In other terms, we assume that the used hash-tags are a description of the location. We are well aware that this might not always be the case (for example, in the case where access

[1] http://www.instagram.com
[2] http://instagram.com/developer/
[3] https://www.yahoo.com/travel/forget-selfies-instagram-actually-makes-traveling-99392573317.html
[4] http://www.jetpac.com/
[5] http://www.polyvore.com
[6] http://www.airbnb

is only possible through wifi and people tag all their photos with the place where they are while posting instead of where they took it). We try to consider this by preprocessing the data set carefully. The report is structured in the following way. We firstly present related work, characterize our extracted dataset, present the algorithm and architecture, followed by results and the conclusion.

3 Related work

For the domain of travel planning, not many related papers have been published so far. J-F. Bérubé et al. [1] solves the travel planning problem by computing time-dependent shortest paths through a fixed sequence of nodes. Their work focuses mainly on optimizing the visit of a given set of nodes in order to see more. Their work divers from ours in the sense that our recommendations are based on only a topic. Yu Zheng et al. [] propose the mining of GPS traces in order to generate two types of travel recommendations. The first type is a generic one, that recommends interesting travel locations and sequences in a given geospatial area. The second type are personalized recommendations matching her travel type. They introduce tree-based hierarchical graphs to model users' location histories and propose a Hypertext Induced Topic Search based model to infer the interest level of a location. For personalized recommendations, collaborative filtering methods are used. The used dataset differs significantly from ours. Compared to GPS traces, traces of instagrams suffer from sparsity. Shuangyu Yu et al. [2] used a Instagram as their data source to recommend Hot Travel Routes. They present a novel concept, called Journey Group (JG), which is a group trajectory pattern reflecting a large number of users who walk through a common trajectory and present a novel Journey Group Trajectory Patterns mining strategy from user generated content based data. Part of their architecture and data cleaning is similar to ours. The processing of clusters differs from ours in the sense that they generated trajectory information based on timestamps and users with multiple instagrams. Our data analysis showed that such an approach can induce a bias. We therefore worked with textual content of instagrams instead.

4 Data Format

The dataset of instagrams has a total size of over 712000 geo-located instagrams out of the environment from Paris, including Saint-Denis, Colombes and Versailles. The data has been captured between the $12th$ of October 2014 until the $10th$ of December 2014. The majority of the data is coming from the center of Paris. A subset of the data and its distribution using Google Fusion Tables and Google maps can be found here[7]. The vast majority of clusters is located in the center of Paris.

The data is retrieved in real-time from the Instagram servers. The Instagram API [3] has adapted parts of the Pubsubhubbub protocol [4], which allows being notified when a new photo was posted in a certain geographical area. Currently, our application does not take advantage of the real-time capabilities, as it requires changes in the infrastructure.

Each instagram has the following attributes:

[7] https://www.google.com/fusiontables/DataSource?docid=1nSFPl50d02v4hrNjWx0Vqp-o5lJzDoZOdXo7L2NS#map:id=3/

1. Id - A unique value that identifies the instagram
2. user_name - the author name of the instagram
3. user_id - the id of the user
4. tags - A set of tags attached by the user to describe the taken photo
5. location_name - users can attach a location name indicating the place where the photo was taken
6. location_lat - geographical latitude where the photo was taken
7. location_lng - geographical longitude where the photo was taken
8. filter - chosen filter for the photo
9. created_time - Date & Time when the photo was posted
10. image_url - Url to the image
11. media_url - Url to the instagram (including likes, text and comments)
12. text - The text attached to the instagram

While all the photos have geographical coordinates (latitude and longitude), only a third of the data contains also contains data for the location_name. We tried to correlated the location_name column to the foursquare database and identified 10826 different venues in Paris that were mentioned in instagrams. For all those venues, we retrieved through the application programming interface (API) of Foursquare additional information, including: the exact coordinates, the address, category, the price level (between 1 and 4) as well as different statistics, such as rating, count of check-ins, likes and tips.

4.1 Exploratory Data analysis

In this section, we describe the basic characteristics of the dataset. Only a subset (84531 instagrams) of the large dataset has been augmented with foursquare data.

4.1.1 Location information

Users can add the location name to their posting. This is quite a popular feature (302164 instagrams have a location_name out of 712485), making up 0.424 percent of the data. Users can either select a place out of a list of locations or create the place themselves. Unfortunately, this introduces some noise and leads to ambiguities, because popular locations exist in different names for different languages. Furthermore, some users abuse the feature to post SPAM and advertisement messages. As stated in section 4, we decided to pre-process and normalize the column location_name by correlating the received location names with another popular venue service called Foursquare[8]. Foursquare has as instagram an API, that allows search for venues given a query and coordinates.

[8]https://foursquare.com/

4.1.1.1 Most mentioned location names The distribution of location names follows a Zipfian distribution, as one can clearly see in the log-log plot in figure 1.

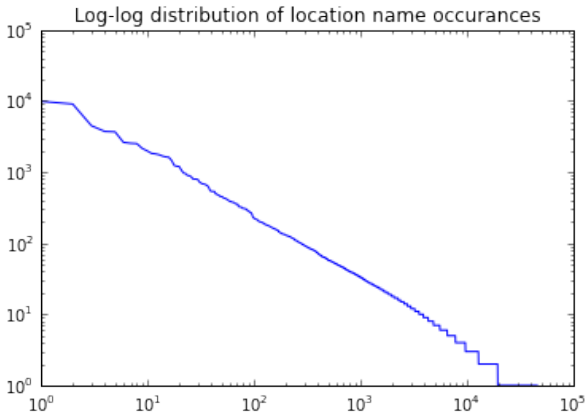

Figure 1: Log-Log plot of rank and frequency of hashtags

Tour Eiffel	12379
Paris, France	9856
Musée du Louvre	9028
Fondation Louis Vuitton	4474
Cathédrale Notre-Dame de Paris	3713
Centre Pompidou	3664
Paris	2607
Château de Versailles	2533
Sacré-Coeur, Paris	2507
Galeries Lafayette	2144
Tour Eiffel - Place Du Trocadero	1982
Eiffel Tower	1826
Jardin des Tuileries	1794
Paris Bercy (Officiel)	1748
Place de la concorde	1671
Montmartre	1628
Jardin du Luxembourg	1610
Champs Elysees Paris	1449
Grand Palais Paris	1232
Avenue des Champs-Elysées	1206

4.1.1.2 Venues The locations that are left over after having pre processed the location_name column are the venues. For all venues, we have additional data, such as the category and price level, exact coordinates and statistics about check-ins. Mostly, we were interested in knowing the distribution of categories of locations, as showed in figure 2. The amount of streets, homes, office and hotels is surprisingly high.

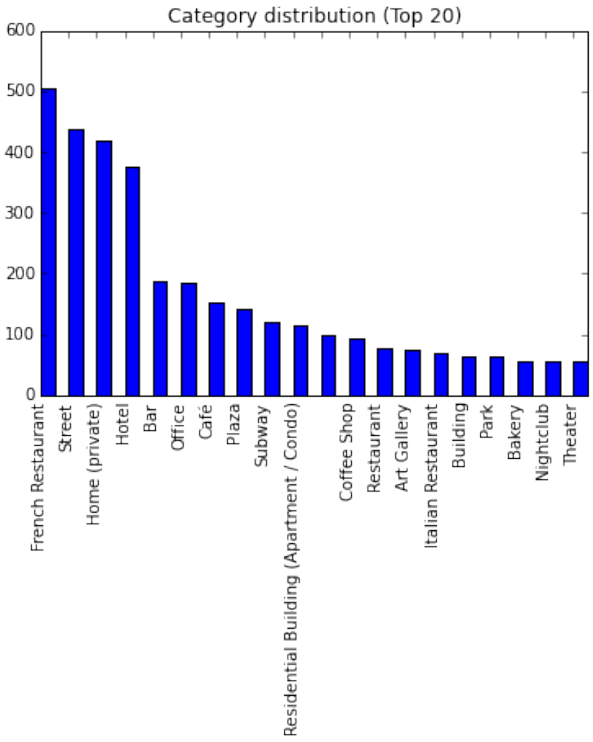

Figure 2: Distribution of categories of locations

4.1.2 User information

In this section, we'll describe the characteristics of users in the dataset. The dataset has a total of 162′972 users. Each user takes on average 4.371825 instagrams. The median is 2, standard deviation is 9.347827. Minimum is equal to 1, maximum equal to 937. The first 10′000 users can be modeled by a power law distribution, as it is observable in figure 3. The low median explains the strong drop after user of rank 10^4. 50% of the users have taken 2 or less instagrams during their trip.

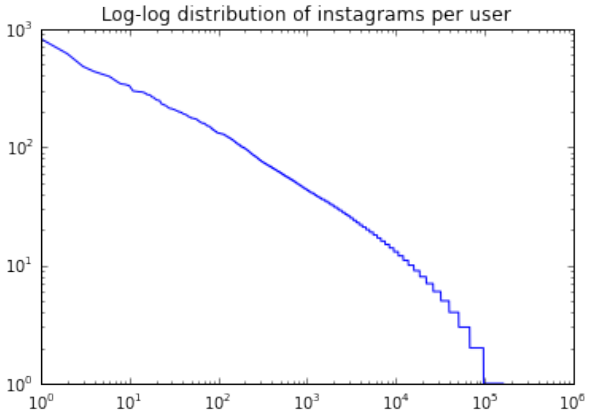

Figure 3: Distribution of instagram's per user

4.1.3 Graph analysis

Based on the correlated location data and the fact that some users have taken multiple photos, we generated an undirected graph. A node is equal to a venue, whereas an edge means that a person has taken an instagram at both places. The weight of an edge is equal to the number of people that have posted an instagram at both locations. As we are using the venue data, only a sample of about 84531 of the original data is used. Interestingly, after building the graph, we find 233 different unconnected components. The main component has 6414 nodes, while the other 232 subgraphs have a maximum of 4 nodes (most of them have only 2 nodes). We discarded all subgraphs but the main component in the graph analysis.

4.1.3.1 Edge distribution and weights There is a total of x edges in the graph. Mean of edge weight is equal to x, median x, standard deviation x, minimum x and maximum x. In figure, one can see the distribution of edge weights. The connected nodes of the top edges are listed in table k.

4.1.3.2 Edge length We were interested in knowing about the length of the edges, meaning the distance in meters between two instagrams taken by a users. We plotted two different figures: In figure 4, every edge is only counted once, while in figure 5, each edge is counted its weight times. From comparison of the two figures, short edges have proportionally higher weights than longer edges. We have the following characteristics: Mean = $3433.18m$, Median = 2586.77, maximum distance = 22153.2, minimum distance = 0.0. The maximum distance is from the center of Paris to Versailles.

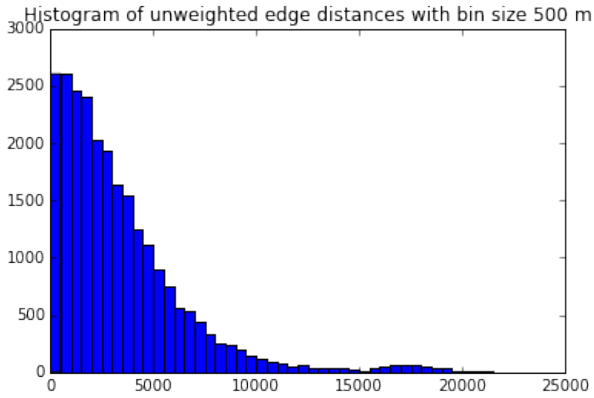

Figure 4: Histogram of edge length. Every edge is counted only once.

4.1.3.3 Degrees of nodes The degree of a node is the number of edges incident to the node **??**. In our graph, the degree of a node is equal to the number of Instagrams taken at a location (multiple instagrams by a user at a location count as one). In figure **??**, once can see that the most popular touristic places in Paris have also the highest degree in the graph.

Edge weight	**Node 1**	**Node 2**
729	Musée du Louvre	Pont Alexandre III — Paris, FRA
502	Tour Eiffel	Le Jules Verne
297	Métro Porte de Versailles [12]	Place de la Porte de Versailles
246	Parvis de Notre-Dame	Musée du Louvre
207	Place du Trocadéro	Tour Eiffel
201	Tour Eiffel	Château de Versailles
190	Bibliothèque des Arts Décoratifs	Les Arts Décoratifs
171	Tour Eiffel	Musée du Louvre
156	paris drobs flat	Les Citadines
149	Salon du Chocolat de Paris	Métro Porte de Versailles [12]
145	Park & Suites Grande Bibliotheque Hotel Paris	Musée du Louvre
140	C.C Saint-Lazare Paris	Café Marco Polo
131	Jardin d'Acclimatation	Fondation Louis Vuitton
128	Le Florence	101 Taipei
110	Café Pasteur	Institut Pasteur
108	Musée du Louvre	Musée National Gustave-Moreau
108	La Paillasse	Carreau du Temple
106	Musée du Louvre	Pont des Arts
104	Arrêt Cité Palais de Justice	Musée National Gustave-Moreau
99	Salon du Chocolat de Paris	Place de la Porte de Versailles

Table 1: Top edges in terms of weights and their associated nodes

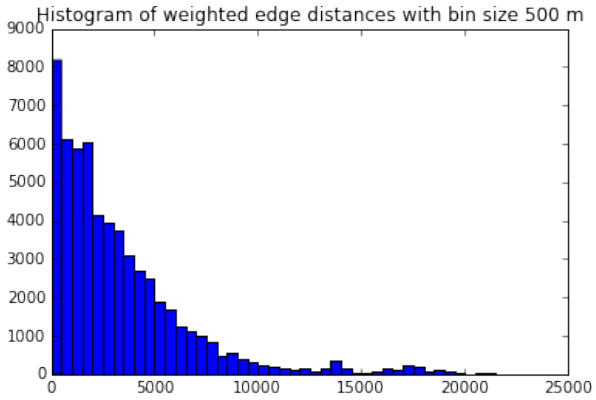

Figure 5: Histogram of edge length. Every edge is counted *weight* - times

4.1.3.4 Betweenness of nodes The betweeness of a node is equal to the number of all shortest paths that go from one node to another node and pass through the the node in question. Betweenness indicates the centrality of a node in the network. A node that has a high betweenness value has a large influence on how people move from one place to the other, under the assumption that they take the shortest path. Again, the locations with the highest betweenness factors are the most well known and the set of top nodes is very equal to the set of top nodes with high node degrees. Figure **??** shows the locations plotted on a map, together with their associated betweenness value.

4.1.4 Communities

We also looked at the structure of communities inside the graph. For this, we used the Louvin Community detection algorithm proposed by V. Blondel and al. [5]. It is implemented for the Python Networkx Package[9] **??**, as well as for Gephi[10]. The outcome was highly interesting: Most of the detected communities were extremely small ($4-5$ locations only), but there were 3 communities that had 321, 398 and 538 locations. After further analysis of the geographical and categorical structure of those big communities, we were not able to find a specific pattern. One community seemed to have many more private homes and local restaurants but less hotels and could represent a community of local and french people. Unfortunately, it would be very difficult to verify this hypothesis. We though of one approach using the language in the comments. Unfortunately, the texts are mostly a set of hash-tags and very difficult to attribute to any language. Therefore, we didn't verify the hypothesis further. Figure **??** shows the histogram of communities (cut at size 100, as there are only a few afterwards).

4.1.5 Hashtags

The total number of different hashtags equals 290588. There are x hashtags per instagram on average, with median equal to *median*. Standard deviation is equal to *std*, maximum to *max* and minimum to *min*. The 20 top instagrams are listed in table 2. From the figure it is observable that many of the most popular hash-tags are general or instagram

[9] https://networkx.github.io/
[10] https://gephi.github.io/

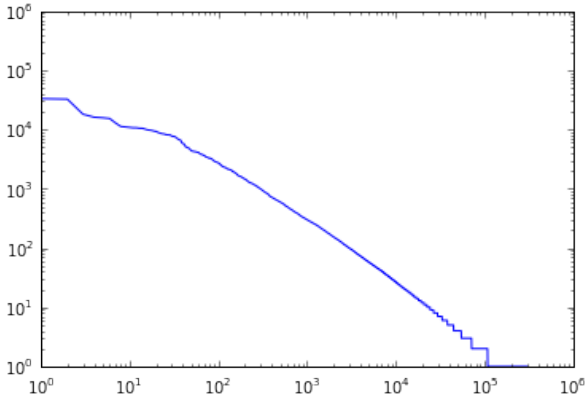

Figure 6: Log-Log plot of rank and frequency of hashtags

specific. As an example, the hash-tag "vscocam" is used by a mobile application called "VSCO" [11], which manipulates photos and posts them on instagram. The distribution is illustrated in as a log-log plot of the rank & frequency in figure 6.

4.2 Conclusion

We finalize this chapter with a conclusion about the exploratory data analysis. We indeed found interesting characteristics of the data that help us further on developing the necessary architecture. We've seen that the median of taken instagrams per users is 2, whereas the mean is over 4. This strong skew is problematic when working with graphs, because: a) The generation of an edge requires at least 2 posted instagrams. Therefore, we can use almost only half of our data. b) Many of the users contribute only 1 or 2 edges, but a couple of users add large amounts of edges. This may induce a strong bias, as the ones that add large amounts of edges are most probably not tourists. We also saw that using community detection algorithm yields large numbers of communities. The modularity factor of 0.405 is medium. This could be interpreted that some sophisticated internal structure does exist. A deeper analysis of the communities could give perhaps additional insights and metrics to classify a location as rather touristic or local. The distribution of edge length is interesting, as most of the edges are very short length, especially because the center of Paris is quite large. We've seen that the number of hash-tags seems very promising. Applying pre-processing on the hash-tags, we could extract a part that follows the natural linguistic characteristics of a corpus and has related semantical meaning with the content. In the following chapter, we will describe how the extracted information has been leveraged to create an application.

5 Architecture

Currently, our architecture focuses on batch processing. In the future, the current architecture could be transformed into a real-time system, using incremental versions of the

[11] http://vsco.co/vscocam

Hashtag	**Frequency**
paris	148190
love	33049
france	32439
instagood	18081
picoftheday	16081
fashion	15723
photooftheday	15293
beautiful	12822
art	11216
friends	11017
tagsforlikes	10831
girl	10678
vscocam	10667
food	10487
happy	10469
followme	10105
instadaily	9843
igers	9723
me	9601
follow	9373

Table 2: The 20 most popular hash-tags in the dataset

currently used algorithms. In figure 7, one can see a simplified schema of the current processing architecture. In the next chapters, we'll detail them out more deeply.

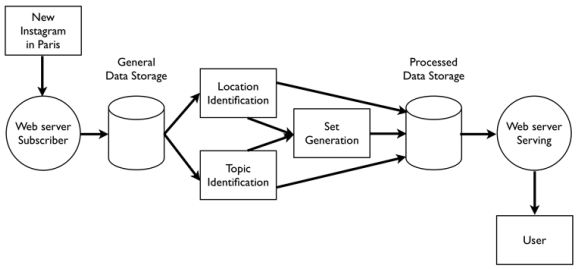

Figure 7: Batch processing architecture of the current #Travel system

5.1 Subscription and Storage

Instagram's API allows the subscription for geographical location. For each publicly posted instagram, the subscriber receives a notifaction from Instagram's server. He can then retrieve the complete instagram using the received identifier. The retrieval is limited to 5000 requests per hour **??**. Our system receives the events, requests the whole instagram and pushes it to a geospatial database. At the time of writing, the main table had a volumne of 700000 instagrams.

5.2 Densitiy-based clustering

The batch processing workflow starts with the location identification. For acceleration purposes, the geographical data is first partitioned using a grid-based approach. We draw a grid of 200 on 200 cells (@TODO: Put distance) over the city of Paris. The data of each cell (and surroundings, in case clusters are on the cell border) can now be processed separately (using distributed computing in case of large data sets). Our system uses the widely known DBSCAN [6] (Density Based Spatial Clustering of Application with Noise) algorithm. A demonstration and implementation of the algorithm is provided by the Python Machine Learning Framework Scikit-Learn [7] and can be found here[12]. This algorithm has two hyperparameters:

1. $MinPts$, which defines the minimum number of points required in a certain area for identifying a cluster. The value depends highly on the size of the data set. In our case, with $> 700k$ data points, we chose it to be around 40.

2. ϵ is a distance value that has to be chosen by the user as well. In our case, we set this value to $12m$

In our dataset, DBSCAN finds a little bit more than 1000 cluster locations. We reduce this number through a post processing sequence to 752 locations. Our post processing sequence has two purposes:

[12] $http://scikit-learn.org/stable/auto_examples/cluster/plot_dbscan.html$

1. Eliminate clusters that have instagrams posted by less than *threshold* users. Plot **??** is a histogram of clusters, binned by the number of active users. We chosen this *threshold* to be equal to $MinPts$.

2. For each identified cluster, we try to assign a name. For this purpose, we make for each cluster a request to the Foursquare API. In case of a success, we retrieve the name, category, image and other information. For about 30% of the cases, we are not able to find a match on Foursquare. Those clusters are eliminated.

The result of the post processing step is stored in a second database for fast retrieval from the serving layer.

5.3 (Biterm) Topic Modeling

For the semantic clustering of our data, we tried different algorithms in the area of Topic Modeling. Conventional topic modeling techniques, such as Probabilistic Latent Space Indexing (PLSA) [8] or Latent Dirichlet Allocation [9] model a document as a mixture of topics. A topic is a set of correlated words, typically represented as a distribution of words over the vocabulary. Using statistical analysis, word distributions for each topic and topic proportions for each document are computed. Conventional topic models reveal topics within a text corpus by implicitly capturing the document-level word co-occurence patterns. Research shows [10] that directly applying these models on short texts, such as tweets, has bad performance due to the data sparsity problem.

Applying topic modeling using Latent Dirichlet Allocation gave us rather bad results. You can see a list of topics coming from LDA in **??**. Due to the lack of performance, we were looking for approaches solving the issue of sparsity. As one can see in figure **??**, the number of hashtags per instagram is relatively low (mean is equal to x). Xueqi Cheng et al. [10] propose a new algorithm called "Biterm Topic Modeling", in short "BTM". Because conventional topic modeling approaches try to model word generation on a document level, the approaches are highly sensitive to the shortness of documents, due to the fact that co-occurence patterns in a single short documents are sparse and not reliable. Therefore, they propose in their paper to aggregate all word co-occurence patterns in the corpus, as these frequencies would be more stable.

A document with three distinct words will generate three biterms:

$$(w_1, w_2, w_3) \Rightarrow \{(w_1, w_2), (w_2, w_3), (w_1, w_3)\}$$

, where the tuples are unordered. The whole corpus turns into a biterm set after extraction of biterms out of each document. Instead of modeling the generation of words in documents, BTM models the generation of biterms. If two words co-occur frequently, chances are high that they belong to a same topic. Xueqi Cheng et al. provide besides the algorithm in the paper also a working version on a Google Code Repository [11]. We integrated their application into our workflow. It takes multiple steps to generate the topics and compute the distribution of topics inside a location. The following list documents the steps:

1. In a first step, documents are cleaned such that words that occur too frequently are removed, as well as words that occur less than 10 times in the whole corpus. A document is equal to the hash-tags posted with an instagram.

2. The BTM algorithm estimates the probabilities of $P(z)$ and $P(w|k)$ using the Gibbs algorithm described in [10].

3. We use the inference algorithm of BTM to infer $P(z|d)$ for each document.

4. For each location cluster C_i computed with DBSCAN described in section 5.2, we compute:

$$\frac{\sum_{p_j \in C_i} p_j}{|C_i|}$$

5.4 Set generation

For each of the extracted topics z, we would like to generate the most adequate set of locations that could be interesting for a user. Besides accuracy, we identified diversity as an important factor for these sets: 5 Art Museums for the topic "art" could be highly accurate, but it would be more interesting if a user could also get other locations in different categories (e.g. Restaurant, Bar) that match the topic.

Algorithm 1 Set computation algorithm

Input: Matrix M with $p_j = P(z_k|l_j)$ for each location l_j and each topic z_k, Category c_j for each location j, metric function f that needs to be maximized
Output: For each topic z, a set of 5 locations

```
for k = 0 to N_z do
  location_index ← 6
  m_k ← sortM[k]
  R ← top 5 locations in m_k
  C ← categories of locations in set R
  best_result ← f(R, C)
  best_set ← R
  while unique categories in C≠ 5 do
    replace ← index, replace_valoflowestvalueincategorieswhere|C_i| > 1
    for index, replace_valinreplace do
      R_i[index] ← location_index
      C_i ← categories of locations in set R_i
      if best_result > f(R_i, C_i) then
        R ← R_i
        best_result ← f(R_i, C_i)
      end if
    end for
    location_index ← incr(1)
  end while
end for
```

For each topic, we would like to generate the most adequate set of locations that could interest a user. We identified diversity as an important factor for these sets: 5 Art Museum for the topic "art" could be highly accurate, but the usefulness is only limited.

Our set generation algorithm should therefore favor topic sets that have a more diverse set of location categories. Our algorithm optimizes the formula:

$$f() = \lambda_{topic} * \sum + \lambda_{category} * \sum_{|C|} 1$$

We can show the two cases. We have set size of l and the values $x_1, x_2, ..., x_l \in C_1$. All values are sorted such that $\forall i, j : x_i > x_j when i > j$. In the case of replacing $x_l with x_k$ where $k = l + 1$ and location $k \in C_1$. We see clearly that the we should not replace x_l with x_k:

$$\lambda_1 * \sum_{i \in 1,...,l} x_i + \lambda_2 * 1 > \lambda_1 * \sum_{i \in 1,...,l-1,k} x_i + \lambda_2 * 1$$

In the case that location $k\ in C_2$, we would have the following condition:

$$\lambda_1 * \sum i \in 1, .., l x_i + \lambda_2 * 1 > \lambda_1 * \sum_{i \in 1,...,l-1,k} x_i + \lambda_2 * 2$$

Location l is replaced by location k if $x_k > \lambda_1 * x_l - \lambda_2$. We can easily see that the formula is easily generalizable.

We set the ratio $\frac{\lambda_{category}}{\lambda_{topic}} = \frac{1}{2}$ and compute the λ's such that in the best case, the top 5 locations are all in different categories. In this case, we set $f() = 1$. In xx one can see the algorithm that optimizes the f in an efficient way.

5.5 Serving layer

The serving layer simply handles requests coming from users. The serving layer has access to the serving database and renders the data into a readable and accessible format to the user (HTML). The serving layer is accessible at $http://www.hashtag-travel.com$.

5.5.1 Application

The interface of the application is relatively simple and consists of mainly two views. The first view 8 allows the selection and discovery of the broad range of topics, whereas the second view 9 focuses on visualizing the different locations of the selected topic.

6 Results

In the following table, we listed the 40 different extracted topics. For each topic, we list the set of 5 locations (with the associated category name).

7 Conclusion and future work

From a student point of view, this project was highly interesting. During the project, I walked through the following points:

1. Think about the necessary data and possible data sources, retrieve enough data

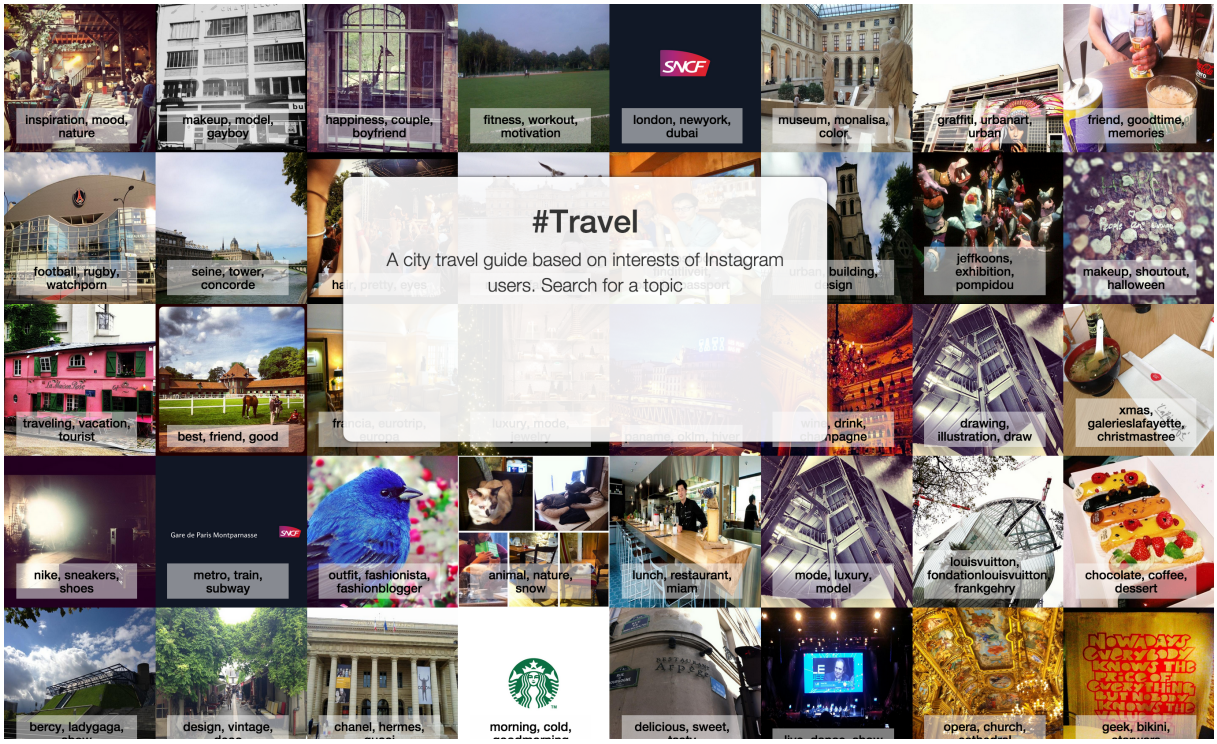

Figure 8: Title page of #Travel

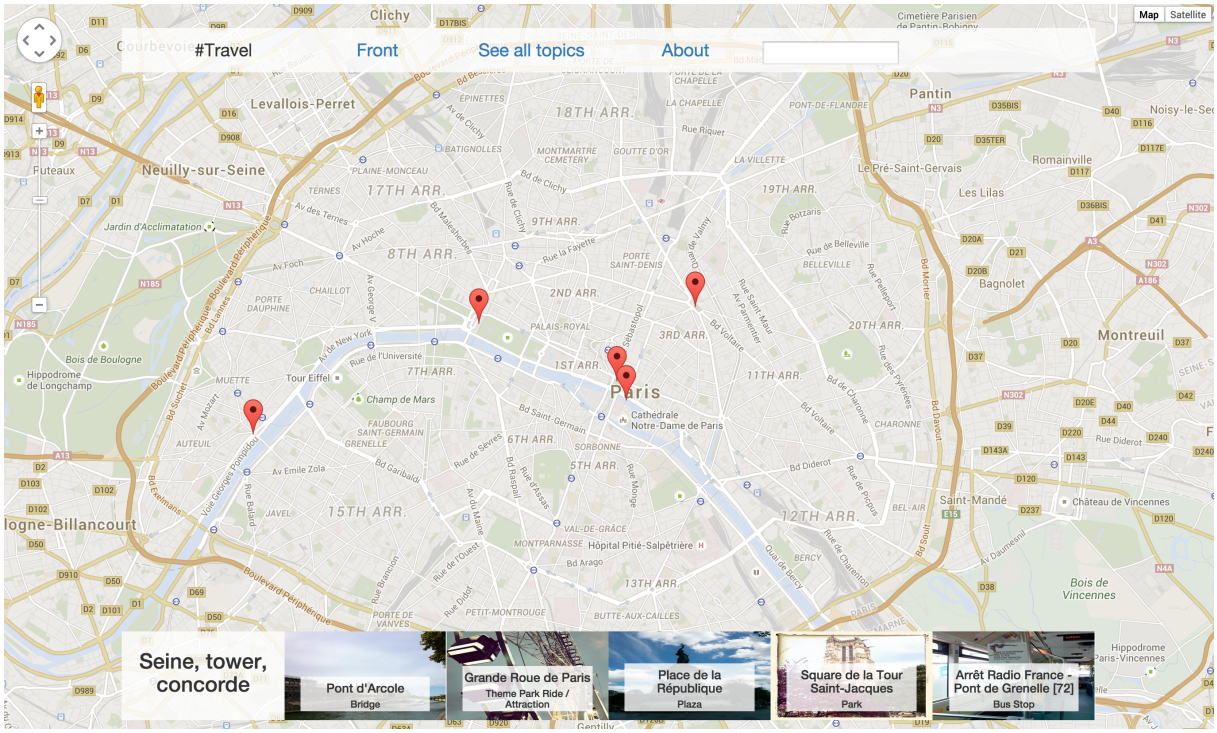

Figure 9: A set of locations for the topic "Seine, tower, concorde"

2. Characterize the data using exploratory data science
3. Apply suitable techniques (DBSCAN, BTM) in order to make the data usable

4. Mash up data with other data sources
5. Engineer a work flow that generates a final result (sets)
6. Build a web application that allows users to interact with the final result

Nevertheless, there is still a lot of work to be done in order to make the workflow more efficient and more suitable to the needs of users. For future work, the batch processing could be ported to real-time, using incremental versions of both DBSCAN and BTM algorithms. As a result, real-time events happening in a city could be detected and suitable lists of locations recommended. A qualitative and quantitative evaluation process of the generated sets should be put in place, such that further improvements of the algorithm can be measured quantitatively. Furthermore, we suggest improvements in the set generation algorithm, taking into account additional features, such as distance and popularity.

References

[1] Jean-François Bérubé, Jean-Yves Potvin, and Jean Vaucher. Time-dependent shortest paths through a fixed sequence of nodes: application to a travel planning problem. *Computers and Operations Research*, 33(6):1838–1856, 2006.

[2] Shuangyu Yu, Yaxin Yu, Yulong Li, and Xin Liu. Poptour: Discovering journey group t-patterns from instagram trajectories to recommend hot travel routes. In *Proceedings of the 2014 IEEE 15th International Conference on Mobile Data Management - Volume 01*, MDM '14, pages 368–370, Washington, DC, USA, 2014. IEEE Computer Society. ISBN 978-1-4799-5705-7. doi: 10.1109/MDM.2014.56. URL http://dx.doi.org/10.1109/MDM.2014.56.

[3] Instagram api. URL http://instagram.com/developer/realtime/.

[4] The pubsubhubbub protocol. URL https://en.wikipedia.org/wiki/PubSubHubbub.

[5] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[6] Martin Ester, Hans peter Kriegel, Jörg S, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[8] Thomas Hofmann. Probabilistic latent semantic analysis. In *In Proc. of Uncertainty in Artificial Intelligence, UAI'99*, pages 289–296, 1999.

[9] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=944919.944937.

[10] Xiaohui Yan, Jiafeng Guo, Yanyan Lan, and Xueqi Cheng. A biterm topic model for short texts. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 1445–1456, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-2035-1. URL http://dl.acm.org/citation.cfm?id=2488388.2488514.

[11] Google code website of btm. URL https://code.google.com/p/btm/.

[12] Yu Zheng and Xing Xie. Learning travel recommendations from user-generated gps traces. *ACM Trans. Intell. Syst. Technol.*, 2(1):2:1–2:29, January 2011. ISSN 2157-6904. doi: 10.1145/1889681.1889683. URL http://doi.acm.org/10.1145/1889681.1889683.

[13] Louvin community detection repository. URL https://bitbucket.org/taynaud/python-louvain.