GuruTravels

A project for the course of *Web & Social Information Extraction*
at *Università di Roma "La Sapienza"*

Romeo Lanzino, 1753403

August 2020, AY2019/20

Abstract

In this report I'll talk about how I've managed to work on this project, explaining how I've built the datasets and how I've accomplished the tasks.

Contents

1 Introduction

The title **GuruTravels** refers to a fictional travelling agency that helps users to find new interesting places to visits based on user's data and avant-guarde machine learning technologies.

All the code in the project has been written in **Python** because of its semplicity and community support: nevertheless, it's the most used language for machine learning purposes thanks to libraries like `sklearn`, `nltk` and `pytorch`.

Most of the code has been executed on my machine, but scraping has been done with a combination of **Google Colab** and **Google Cloud**'s machines because of their speed of execution and connection, fundamental for our purposes.

2 Scraping the data

For this job, we are given two datasets retrieved from Gowalla and Brightkite, two dead social networks about travelling.

Data can be summed up in **users**, **friendship relationships** and a list of **places** visited by users.

As we were encouraged to build or find our dataset by ourselves, I've decided to build one myself scraping informations from a social network.

2.1 Choice of the social network to scrape

2.1.1 Instagram

Initially, my choice fell on Instagram as it's a "young" social, very populated and well-known, based not on locations but on pictures that can, optionally, have a location tag in their description.

But Instagram has many cons:

- **does not have APIs** for non-premium users
- **limits the number of requests** even if the user is using a browser, without telling when he can visit other pages
- it's **a nightmare to scrape**, as each page has Javascript in it (we couldn't simply request the HTML of the page without a browser) and tags' attributes are named randomly, as in figure 1

After these considerations, I've decided to discard the choice of Instagram in favour of another social network, although partially-working code made with Google Colab is still present (but unused) in project's folder.

2.1.2 Trover

My second choice fell on Trover, a not-so-popular social network based on landscapes pictures, where each photo is tagged with a location and users can follow

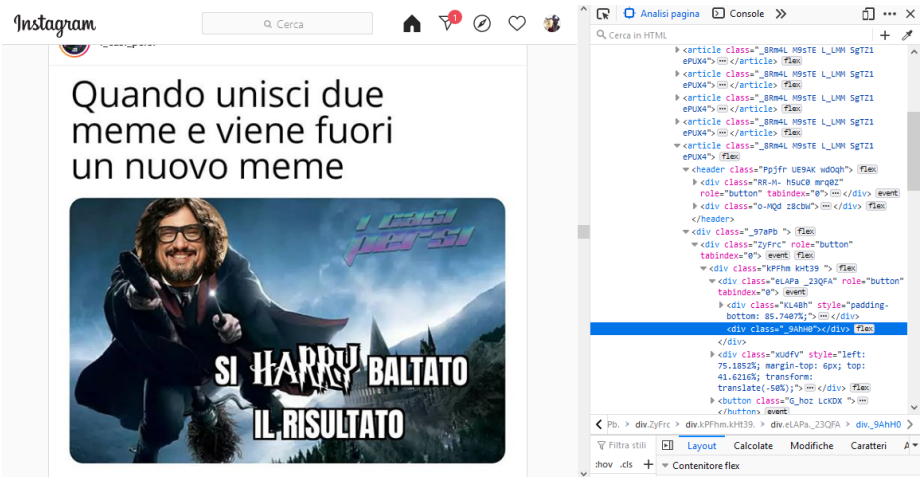

Figure 1: Example of Instagram's HTML relative to a picture

other users just like in Instagram: perfect for our needs.

The problem with Trover is that's **not very popular (not so many users)** and **scraping was not so fast (although easy)** because I had to implement a browser driver to scroll down the pages in order to reach all the followers and the locations visited by each user: a process that took from 10 to 15 seconds per user.

2.1.3 Foursquare

My final choice has been Foursquare, a user-friendly website, service and social network used by companies like Uber, Coca-Cola and Twitter because of its **APIs**, used by me (with a public token) to reach $101K$ users in just two days of scraping.

2.2 Scraping process

The scraping program was entirely done using **Foursquare's exposed public APIs** through **HTTP requests** made in Python; code was then executed on a machine hosted **Google Cloud** services with $0.6GB$ of RAM but an astonishing $200Mbps$ download speed, in order to avoid my home's $7Mbps$ ADSL.

As better explained in section 3.1, each user is uniquely identified by an alphanumeric **user-id**: in analogy with webpages URLs, I've built a simple crawler that starts with a list of seeds (user ids) and does a DFS on their followers and followees, adding them to the frontier with a FIFO policy.

The code for the crawling of the various websites can be found inside the main folder.

3 Representations

Data is populated by two distinct types of entities: **users** and **locations/pois** (points of interest).

3.1 Users

Users are scraped with these infos:

- `id`: unique string that identifies the user
- personal infos:
  - `first_name`
  - `last_name`
  - `gender`
  - `home_city`
- `pois`: list of points of interest `id`s visited by the user
- `followings`: set of users followed by the user
- `followers`: set of users that follows the user

A user is then represented in memory as a **weighted vector of interests**, where each position represents a category of POIs (e.g. "Country club", "Pub", "Historical monument") and its value the normalized amount of times the user has visited a POI with that category.

Let's say that the recommender system has a vocabulary of 10 interests and a user u has visited two POIs that holds categories 1 and 3. The user is then represented as:

$$u = \left[0, \frac{1}{2}, 0, \frac{1}{2}, 0, 0, 0, 0, 0, 0\right] = [0, .5, 0, .5, 0, 0, 0, 0, 0, 0]$$

Even if those vectors are rather sparse, all their values will sum to 1 in order not to prefer more social users.

u is then concatenated to a flag (1 or 0 values) representing the gender.

Using categories of POIs rather than POIs or cities themselves reduces the dimensionality of the vectors, that at the end is ≈ 480.

3.1.1 POIs

- `id`: unique string that identifies the point
- `name`: name of the POI
- `categories`: set of categories/interests of the POI (e.g. "Restaurant", "Barbecue")
- `address`:
  - `address`
  - `zip`
  - `city`
  - `country`
  - `latitude`
  - `longitude`
- `stats`:
  - `rating`: average $\in [0, 5]$ of votes given by users to the POI
  - `popularity`: $\in [0, 1]$
  - `checkins_count`: number of recorded visits to the place

Rather than representing POIs, it's much better to represent cities: each city c is modeled just like users' vectors, looking at POIs categories in town.

c is then concatenated to two normalized values representing the average of ratings and the popularity, allowing to raise $F1$ scores by an astonishing 10%.

4 Simpler tasks

In order to solve those simpler tasks, users are represented as nodes of a directed graph and connected based on followings and followers of each user.

The documented and self-explanatory code can be viewed in `assets/tasks.py`.

4.1 Similarities

Jackard and cosine similarities are shown in files `assets/jackard_similarities.csv` and `assets/cosine_similarities.csv` according to project's text.

Code is self explanatory, but in particular for cosine similarities I've used Python's `numba` package to parallelize the computation on a GPU in order to gain a significant speed-up. Also for cosine similarities, all calculus are done using an algebraic approach using `numpy` package.

4.2 Density

Densities of the most strongly-connected components are shown in file `assets/sccs.csv`.

To find those components I've implemented an iterative version of **Tarjan's algorithm** that does a single DFS to find all components.

Finally, since this is an ordered graph, formula is modified from $D = \frac{2|E|}{|V|(|V|-1)}$ to $D = \frac{|E|}{|V|(|V|-1)}$ because for directed simple graphs the maximum possible edges is twice that of undirected graphs to account for the directedness, so we can discard the 2 at the nominator.

However, most of the strongly connected components are formed by just two nodes and have maximum density 1.

4.3 Top brokers

The problem is solved with my implementation of the **KPP-NEG algorithm** that, in brief, works by computing a connectivity measure (in our case *reach* as defined in course's slides) of variants of the original graph prived of each user in turns.

Results for 10 components are saved into `assets/top_brokers.csv`.

4.4 Spread of influence

For the conformation of the graph, the initial 20 infected users couldn't infect anyone after 100 iterations.

Results are shown into `assets/spread_of_influence.csv`.

5 Recommender system

5.1 Overview of the model

The `GuruRecommenderSystem` is a deep neural network made with `pytorch` that solves a **classification problem**: it takes as inputs the representations of a user and a city (POI) and predicts if the user will like it based on its interests and the attractions the city can offer.

To such representation of user and city is concatenated to their **cosine similarity**, offering the net a raw idea of similarity between them and boosts the results by 8/10%.

The model is composed by **two hidden linear layers** interspersed with a **ReLU** activation function and a **dropout** with $p = 0.1$ throughout the whole net. The loss function then uses the **SoftMax** activation function to compute losses and gradients for the backward pass.

The net is then **trained** on a training set that's a 60% partition of the original scraped users for 3 epochs and an average of 8 minutes per epoch.

As you can see in 5.2.2, such solution can even achieve 0.9 of $F1$ on the test set, more than other tried pre-made `sklearn`'s solutions like Random Forests

(because data is not discrete), Multilayer Perceptron (much simpler net that underfits), SVM and Naive Bayes.

5.2 Evaluation

Evaluation has been done both using k-fold cross-evaluation, according to project's text, and using standard metrics on the three sets to check for overfitting and eventual anomalies in performances.

5.2.1 k-fold cross-evaluation

According to project's specification, a k-fold cross-evaluation with variable k has been performed; parameters k has been chosen from a minimum of 5 to a maximum of $|i_{min}| - 1$, where i_{min} is the set of interests expressed by a user (pruned to 15, meaning that each user has expressed at least 15 interests, amounting to nearly $10K$ users).
Each iteration is then divided in k splits, where training (60%), dev (20%) and test (20%) set are chosen at random between the list of users.
A new recommender system is then trained for each split, and for each user in test set a total of k negative interests are chosen between the total list of interests.
Each user now have this list of $2k$ interests (k positives and k negatives) from which a number $1 \leq k' \leq 2k$ are fed to the recommender system's neural network and guessed.
The score of the split equals $\frac{k'_{true}}{k'}$, where k'_{true} is the number of interests correctly identified by the recommender.
Table 1 shows a brief recap of the results from the various splits (more accurate scores on each split can be viewed in file `assets/kfold.json`).

k	worst split's score	best split's score	average of splits' scores
5	0.787	0.816	0.806
6	0.791	0.801	0.798
7	0.778	0.802	0.796
8	0.789	0.802	0.795
10	0.77	0.811	0.787
11	0.758	0.8	0.787
12	0.765	0.789	0.773
14	0.754	0.793	0.77

Table 1: Scores obtained during k-fold cross-evaluation

5.2.2 Non-k-fold evaluation

The set $users$ with $|users| = 101188$ is initially pruned getting rid of users with less than one interest, for a total of $|users| = 54560$, and then partitioned in $users_{train}$, $users_{dev}$ and $users_{test}$, where $|users_{train}| \approx 0.6 \cdot |users|$ and

$|users_{dev}| \approx |users_{test}| \approx 0.2 \cdot |users|$.
Then, a recommender rec is trained on $users_{train}$ for 3 epochs and evaluated on $users_{dev}$ and $users_{test}$ to obtain results in terms of **accuracy**, **recall**, $F1$ and **AUC** as shown in table 2 and image 2.

set	accuracy	recall	$F1$	support
$users_{train}$	0.9	0.89	0.9	635048
$users_{dev}$	0.89	0.89	0.89	209316
$users_{test}$	0.89	0.89	0.89	217518

Table 2: Classification report of the train, dev and test sets

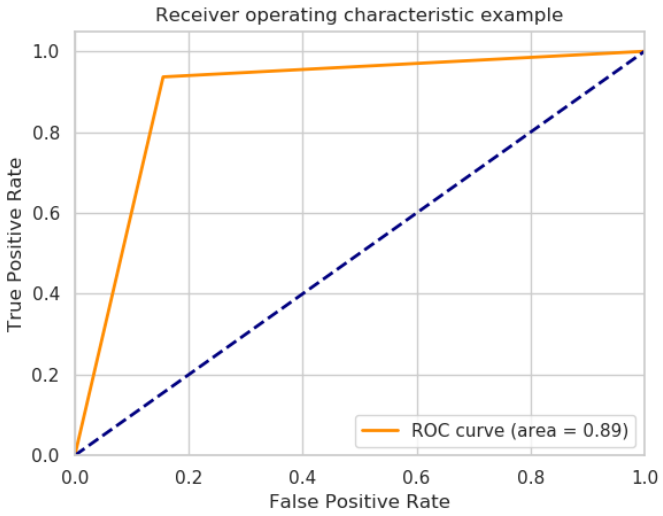

Figure 2: ROC of the evaluation on $users_{test}$

6 Conclusion and future works

There are many approaches for recommender systems, but nowadays the most prominent is the one using deep learning, thanks to the advances in this field.

For example a real agency can be shown just some pictures shot by the client in order to classify them and suggest the customer similar places based on the mood of the pictures shown, or the user could provide a series of places visited in its life in order to feed the time series to a BiLSTM and find a recurring trend in user's tastes and suggests new places to vist.

This report just scratches the surface of this broad field, but the advances that can be made are countless and the limit is just the imagination.