Machine Learning Engineer Nanodegree

Capstone Project

Raphael Roullet
December 1st, 2016

I. Definition

Project Overview

This project would be part of the domain of recommender (or recommendation) systems.
Recommendation systems are used to suggest items to users that they will like, based on different factors, increasing the chances that they perform an action with it (buying, watching…). With the increase of data on the web and the so-called long-tail, recommendation systems have become widely used to recommend different things: items on Amazon, news articles on Google News and music on Spotify…
Two main types of recommendation systems exist:

- Content-based: These algorithms use characteristics of items to make predictions
- Collaborative filtering: This type recommends items only based on the users past behavior. It is based on the principle that if someone with similar tastes than me has liked something, I'm more likely to like it as well.

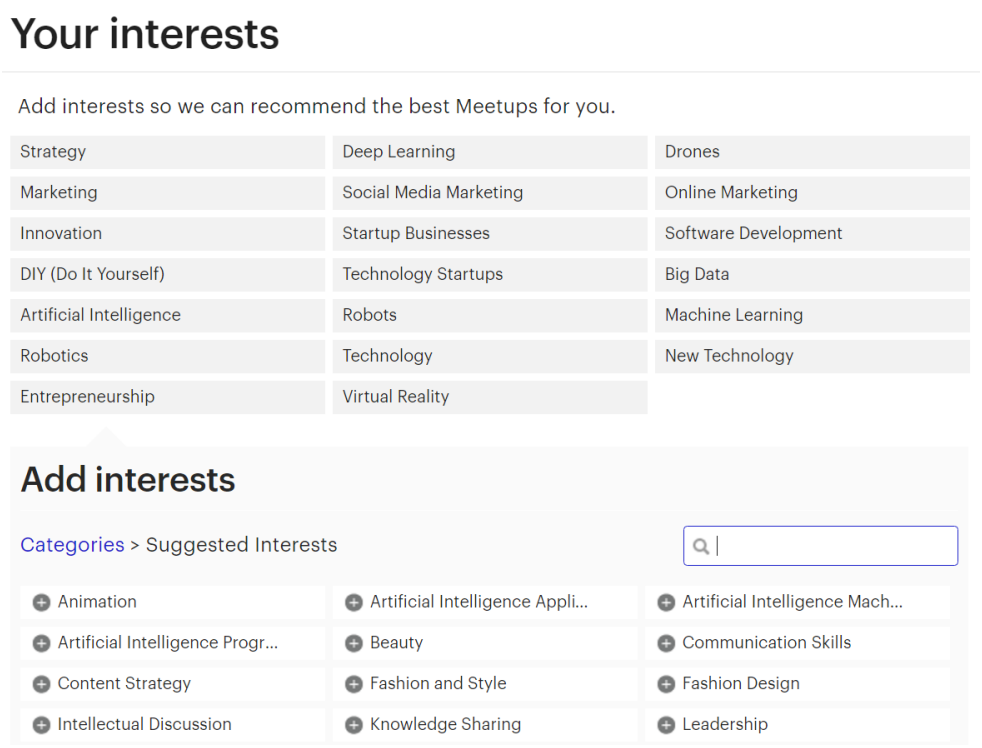
Other types of recommendation systems exist and some, like Netflix's, can take a hybrid approach.

This project however would take a different approach by *recommending users to other users* based on their similarity, a step that is part of collaborative filtering. Nevertheless, it would take a content-based approach by looking at the characteristics of the users, namely their interests. This project, indeed, aims at **recommending to users other users with similar interests, hobbies, passions…**

My personal motivation for this project stems from a desire to turn this project into an API that could be used through a mobile app that would also include messaging capabilities between users.

The ideal dataset would contain data about real people and their interests. We would need a list of users and for each one a list of their interests such that users would have interests in common. The bigger the dataset, the better since algorithms tend to perform better and better when given larger and larger datasets.

It turns out that this data can be found using the Meetup API. "Meetup is an online social networking portal that facilitates offline group meetings in various localities round the world." On the platform, users can add interests to their profiles so that Meetup can recommend Meetup groups to join:



Using Meetup's official API, 103,729 profiles of Meetup members were fetched in a 15-mile radius from Miami. Among these, **82,186 persons** had at least one interest indicated in their profile.

This data seems to be perfect to work with in order to solve our problem since it meets all the requirements mentioned above. It might be needed to ignore persons that don't have at least a minimum number of interests indicated. In addition, only a subset of the 82,186 might be used because of constraints in terms of computational resources.

Problem Statement

Given a dataset of X individuals and for each individual a list of n discrete interests from a finite universe of N interests, the problem is the following:

For a given individual how can we order the list of all the others X-1 individuals in descending order by similarity of interests.

In order to recommend to users the 5, 10 or 20 users with whom they share the most interests we need to compute the similarity of their interests with those of all other users and then rank users by descending similarity. By considering each list of users interests as documents the problem becomes one that appertains to the semantic analysis field which is well-documented.

Metrics

Similarity can be tricky to define as it can be considered somehow subjective. However, we need a quantifiable way to evaluate and benchmark our model.

Based on the problem we are trying to solve we can say that for a given queried user i and a given number of "matches", we want to return matches that share a maximum number of interests with the queried user. In that case, the metric to use for a matching user j seems logically to be the number of shared interests between i and j.

However, based on our data set, we will see in the following section that users have different number of interests listed. Thus the results from our algorithm will be significantly different: one user with 3 interests can only have maximum 3 shared interests with someone else whereas someone with 45 interests might be matched to people with +30 shared, but that doesn't mean that those results are more accurate. So we need to normalize if we want to have a meaningful metric when comparing the results of users with very different numbers of interests listed. The following metric which can be expressed as a percentage seems more appropriate:

$$\Xi = \text{Number of shared interest between i and j} / \text{Number of interests of i}$$

Additionally, by looking at the metric Ξ across several iterations with several random users and by outputting, let's say the top 10 matches, we might want to pay attention to the standard deviation of Ξ for the 10 matches. The standard deviation is the square root of the variance which is the average of the squared differences from the mean. A standard deviation too high would mean that the users share very different percentage of interests with i (for instance 70%, 15%, 40%, 5%, 90%...) and thus reveal that there is a

problem with our algorithm. So after computing matches for X users, the average of the X standard deviations of Ξ can be useful to compute as a second metric.

Other metrics such as Mean Squared Error or Mean Absolute Error which are frequently used with algorithms that make predictions could have been used as they reveal the difference between what is predicted and the true value of what is estimated. Though, the challenge that arises from our problem and evaluating the results of our algorithm is that we don't know in advance what the true values of what is estimated are, i.e the lists of other similar users for each user.

II. Analysis

Data Exploration

The dataset used for this project consists of profiles fetched from Meetup.com using Meetup's API. First, it returned 1266 groups in a 15-mile radius from Miami. We then extracted 103,729 profiles, all members of these groups. Each data entry has the following features, selected among those provided by the API (http://www.meetup.com/meetup_api/docs/2/members/). Here's an example for someone who really seems to like dogs.

```
{
'city': 'Miami',
'topics':
      [
      {'name': 'Pets', 'urlkey': 'pets-animals', 'id': 53052},
      {'name': 'Dogs', 'urlkey': 'dogs', 'id': 15067},
      {'name': 'Active Dogs', 'urlkey': 'activedogs', 'id': 9772},
      {'name': 'Animals', 'urlkey': 'animals', 'id': 37663},
      {'name': 'English Bulldog', 'urlkey': 'engbulldog', 'id': 560},
      {'name': 'Off-Leash Dog Recreation', 'urlkey': 'offleash', 'id': 975
      3},
      {'name': u'Pug', 'urlkey': 'pug', 'id': 591}
      ],
'link': 'http://www.meetup.com/members/111111111',
'id': 111111111,
'name': 'John Doe'
}
```

As seen above, each profile contains information about:

- **City**: Where the person is located
- **Topics**: What we will call interests. For each interest we have the name, urlkey (used to access it on Meetup.com) and unique id of the topic/interest.
- **Link**: towards the person's profile on Meetup.com
- **Id**: Unique id of the person
- **Name**: Name of the person

Among all the information provided for each profile, only topics will actually be used to feed our algorithm. The rest is extra information that could be used for future reference.

Here are the first 10 profiles:

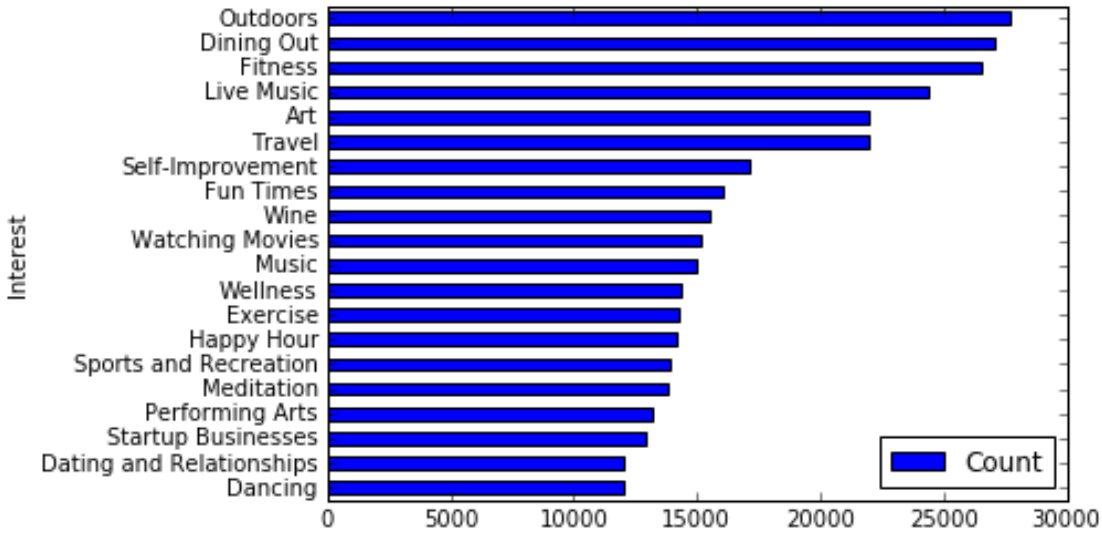
	city	id	link	name	topics
0	Miami	1837039	http://www.meetup.com/members/1837039	...Adeela!	[]
1	Miami	50171992	http://www.meetup.com/members/50171992	"Jay" - Jennifer	[{u'name': u'Black Professionals', u'urlkey': ...
2	Miami	203909843	http://www.meetup.com/members/203909843	Aaron Wallace	[{u'name': u'Pets', u'urlkey': u'pets-animals'...
3	Minneapolis	145409602	http://www.meetup.com/members/145409602	abby	[{u'name': u'Pug', u'urlkey': u'pug', u'id': 5...
4	Miami	201741842	http://www.meetup.com/members/201741842	abi	[{u'name': u'Artists', u'urlkey': u'boston-art...
5	Miami	8484098	http://www.meetup.com/members/8484098	ADOLFO ROBIOU	[]
6	Miami Beach	5694840	http://www.meetup.com/members/5694840	Aimee	[]
7	New York	193427043	http://www.meetup.com/members/193427043	Alan Etienne	[{u'name': u'NFL Football', u'urlkey': u'nflfo...
8	Miami Beach	32100452	http://www.meetup.com/members/32100452	Alan Rubio	[]
9	Miami Beach	10645541	http://www.meetup.com/members/10645541	Alana and Max	[]

From the first 10 entries we notice some discrepancies. Even though in our request to Meetup's API we queried only groups in a 15-mile radius around Miami, we notice some

users, members of these groups, have a different city indicated in their profile. However, this doesn't interfere with our problem solution.
More importantly, we note that some users don't have any interest indicated on their profile. Therefore, they cannot be used for this project.

Exploratory Visualization

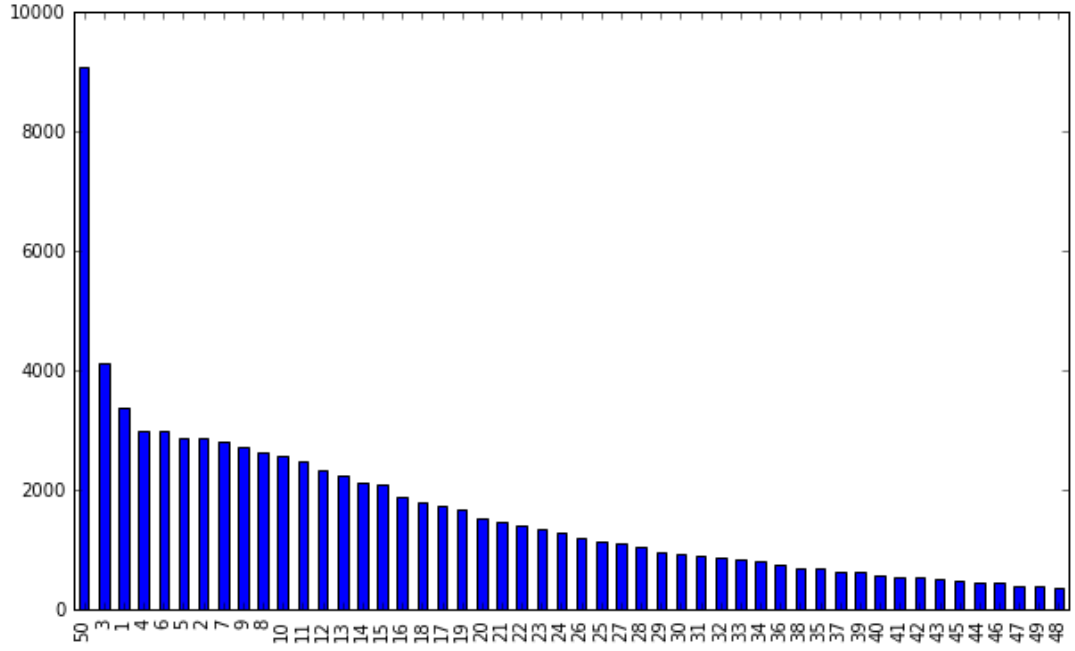
Here are the most common interests among our dataset with their respective count:



The top interests are not surprising: Outdoors, dining out, Music, Art, Travel, Fun times... are all very broad categories that are highly appreciated by most people.

There is a total of 21,892 unique interests.

How many interests do people have indicated on their Meetup profile? The following chart provides a good overview of the breakdown:



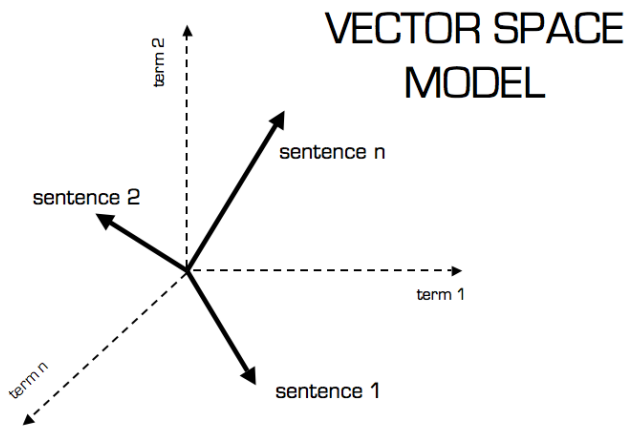
50 is the maximum number of interests you can select on Meetup and by far the most common number among our dataset followed by small number 3,1,4,6… up to 48 in almost ordinal order.

Algorithms and Techniques

Bag of words model

From the Meetup data as input, the approach taken is that of using the **Bag of Words** representation which consists of treating text documents or speeches as a list of words, only keeping the most important words and their frequency but discarding their original order and grammar.

The machine learning algorithms in text analysis (or “text mining”), are designed to work with numerical feature vectors of fixed length rather than documents of variable length made of strings (i.e text). The general process of turning a collection of text documents into numerical feature vectors in a vector space is called vectorization. Here’s what a vector space looks like:



With scikit-learn we apply the following techniques to vectorize our lists of user interests:

- **Tokenizing:** Often times in text analysis, the text documents need to be processed in order to remove common words (a, the, and …), punctuation and only keep relevant words (called "tokens"). Our data in that sense is already cleaned since they are only lists of interests (all of them being tokens). So we create a custom tokenizer whose role is only to return the lists of interests from our data set.
- **Counting** the occurrences of tokens in each document.
- **normalizing** and weighting with diminishing importance tokens that occur in the majority of samples / documents. Weighting will be discussed below.

That way, each individual interest occurrence frequency is treated as a feature and the vector of all the token frequencies for a given document is considered as a sample. We thus obtain a term-document matrix with one column for each unique interest and one row for each user.

This matrix is a sparse matrix which means that most elements are zero. That is easily understandable since users have maximum 50 interests listed and there are 21,892 unique interests resulting in a very high number of non-listed interests (marked as 0 in our matrix).

When analyzing corpuses, it is very common to use TF-IDF. TF-IDF stands for Term Frequency Inverse Document Frequency. "The TF-IDF weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document (TF) but is offset by the frequency of the word in the corpus (IDF).

- The first term computes the normalized Term Frequency (TF), aka. the number of times a word appears in a document, divided by the total number of words in that document;
- The second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

The TF-IDF weight is the product of these two terms." If TF-IDF is used, those weights are the value present in our term-document sparse matrix.

However, in our case, term frequency is not relevant because each interest appears only once in each document. Inverse document frequency seems to be more interesting to explore since it would give a lower weight to interests that most people share and that by definition are not helping us better match people with similar interests.

With 21,892 unique interests, it might be interesting to try to reduce the vector space into a space of lower dimensionality. This can be done by applying Latent Semantic Analysis (LSA) to our term-document matrix. LSA is a technique used to analyze the relationships between a corpus of documents and the terms they contain by producing a set of concepts related to the documents and terms. If we take a simple example with a document represented in a 3-dimension space, after using LSA, this document might be described in a 2-dimension space as follow:

{(car), (truck), (flower)} ↦ {(1.3452 * car + 0.2828 * truck), (flower)}

Latent semantic Analysis would be interesting to perform as it might reveal some interesting associations between interests.

Finally, given a sparse vector X, from our term-document matrix and representing the interests of one person, the challenge is to find the most similar vectors (i.e, user) among all the others. To do this, we use cosine similarity.

"The cosine similarity between two vectors (or two documents on the Vector Space) is a measure that calculates the cosine of the angle between them" and is given by this equation:

$$\cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\|\|\vec{b}\|}$$

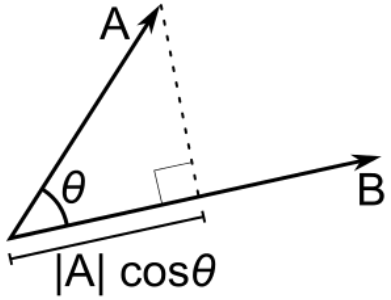
The numerator is the dot product of two vectors $\vec{a} = (a_1, a_2, a_3, \ldots)$ and $\vec{b} = (b_1, b_2, b_3, \ldots)$, where a_n and b_n are components of the vector, associated in our case with interests. The dot product is a single value (i.e a scalar) and defined by:

$$\vec{a} \cdot \vec{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

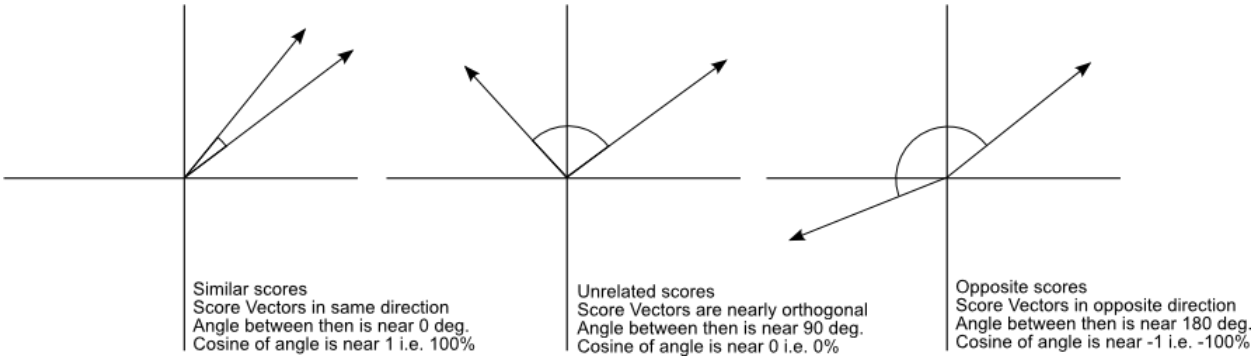
The equation of the cosine similarity can be rearranged into the following:

$$\vec{a} \cdot \vec{b} = \|\vec{b}\| \|\vec{a}\| \cos\theta$$

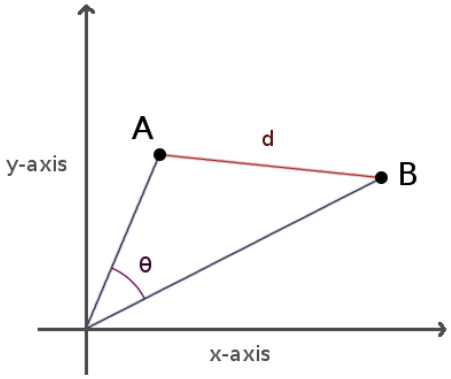
The term $\|\vec{a}\| \cos\theta$ is the projection of the vector $\vec{a}$ into the vector $\vec{b}$ as represented by this image:



The following images help to provide a better understanding of how the cosine similarity changes based on different vectors:



Another measure of similarity between vectors that can be explored is Euclidean distance. Cosine similarity takes the directions of the vectors into consideration, whereas Euclidean distance compares their magnitude which is influenced by how many times the same words appear in a given document. Cosine similarity leaves aside the fact that a document might be three times longer that another one when computing similarity, Euclidean distance does not. Here's a visual example of Euclidean distance, d, in a two-dimensional Euclidean space:



After having computed the similarity between a given vector representing a user and all the others, we sort the vectors by descending order of "similarity score".

Benchmark

As a benchmark model we take the results obtained by using "gensim", an excellent Python library made by Radim Řehůřek, which was designed to automatically extract semantic topics from documents: https://radimrehurek.com/gensim/index.html

Gensim can also be used to compute document similarity and output a list of documents ordered by similarity if given a query *or another document as a query* through its Similarities method that we proceed to implement: https://radimrehurek.com/gensim/tut3.html

It includes its own implementation of tfidf, LSA and cosine similarity. So, we first create a gensim corpus from the exact same corpus (our dataset of users interests).

Given that both the benchmark and solution model were given the same input, we will pick random users and compare what the two models return as the most matching users, i.e the vectors with the highest cosine similarities.

Here's an example of output from the benchmark model with user #678:

```
[ (9499, 0.54554486274719238),
  (6280, 0.53665632009506226),
  (74350, 0.53665632009506226),
  (11039, 0.52704644203186035)]
```

This is a list of the top 4 tuples containing user index and similarity score ranked in decreasing order of cosine similarity.

III. Methodology

Data Preprocessing

A couple of preprocessing steps were taken with this dataset.

From the Data Exploration section, we noticed that some users had no interest indicated in their profile so we remove these entries which can't be used for this project. Among all the profiles, we're left with 82,186 of them who had at least one interest.
I found out that it's very important to re-index our dataframe after removing these entries in order to avoid having empty rows.

The list of interests for each user contains additional information that is useful but unnecessary for our algorithm like so:

```
[{u'id': 242, u'name': u'Fitness', u'urlkey': u'fitness'},
 {u'id': 713, u'name': u'Dining Out', u'urlkey': u'diningout'},
 {u'id': 1998, u'name': u'Travel', u'urlkey': u'travel'},
 {u'id': 8652, u'name': u'Live Music', u'urlkey': u'livemusic'},
 {u'id': 9696, u'name': u'New Technology', u'urlkey': u'newtech'}]
```

We create an additional column in our dataset that only contains the names of the interests and that we consider as "bags of words".

Implementation

We first implemented from scikit-learn a vectorizer that also included a TFIDF processor called TfidfVectorizer in order to extract all interests as features and turn our dataframe of "documents" into a sparse matrix.

One little change was about writing a custom tokenizer. Since vectorizers are normally used to process real documents or transcript from speeches they're designed to extract tokens from sentences and ignore stop words. In our case each user has a simple list of interests that is already tokenized. Hence, our tokenizer only has to return this list without manipulating it.

With the sparse matrix now constructed, we use cosine_similarity from sklearn.metrics.pairwise in order to compute similarity between rows representing users. We create a function that takes as input the sparse matrix, the queried user and the number of top similar users we want to be returned.

Because of computational limitations, I realized that creating a matrix of similarities between for all users against all users all at once was making my computer crash. I

chose to tweak my function to take a user as a parameter and only compute similarity between this user and all the others:

$$sims = cosine\_similarity(X[index,:],X)$$

The other tricky part was to transform the array of similarities, sort it and retrieve the user indices from it. Fortunately the use of np.argort came handy to do all of this at once.

The function returns a list of the top n users among all the others by descending similarity of interests, with index and “similarity score”, providing a good initial solution. However, after creating a function in order to display the interests of the users in that list and comparing them manually a need for improvement came to light.

2 interesting things appeared from the analysis of the list of users similar to user 4325:

```
Match #1
Match score with 56090: 1.0
Interests of 56090: [u'Fitness', u'Vegetarian', u'Vegan']
Number of shared interests: 3 (100% of all 56090's interests)
Shared interests: set([u'Vegetarian', u'Vegan', u'Fitness'])

Match #2
Match score with 55995: 1.0
Interests of 55995: [u'Fitness', u'Vegetarian', u'Vegan']
Number of shared interests: 3 (100% of all 55995's interests)
Shared interests: set([u'Vegetarian', u'Vegan', u'Fitness'])

Match #3
Match score with 55774: 0.952360441362
Interests of 55774: [u'Vegan', u'Vegetarian']
Number of shared interests: 2 (100% of all 55774's interests)
Shared interests: set([u'Vegetarian', u'Vegan'])

Match #4
Match score with 17375: 0.952360441362
Interests of 17375: [u'Vegetarian', u'Vegan']
Number of shared interests: 2 (100% of all 17375's interests)
Shared interests: set([u'Vegetarian', u'Vegan'])

Match #5
Match score with 55823: 0.864171025314
Interests of 55823: [u'Vegetarian', u'Vegan', u'Spirituality']
Number of shared interests: 2 (67% of all 55823's interests)
Shared interests: set([u'Vegetarian', u'Vegan'])

Match #6
Match score with 56001: 0.835556215693
Interests of 56001: [u'Fitness', u'Vegetarian', u'Vegan', u'Organic Foods']
Number of shared interests: 3 (75% of all 56001's interests)
Shared interests: set([u'Vegetarian', u'Vegan', u'Fitness'])
```

The 5th matching user has a score of 0.86, significantly lower than the 3rd and 4th matching users (with a score of 0.95) even though the 5th matching user has all their interests (namely "Vegeterian" and "Vegan"). This means that the algorithm penalizes having additional non-shared interests in terms of similarity. This might be a far-fetched

deduction. What if the queried person simply forgot to indicate that she also likes Spirituality? What's more what if the 4th match forgot to indicate that she likes Hiking, which the queried user hates?

Even more worrisome is the fact that the 6th matching user has *all* the interests of the queried user but is only in the 6th position after users with only 2 matching interests! This is a poor performance if we take the number of shared interests as a metric to evaluate our algorithm.

Refinement

Using a different vectorizer

Sometimes less is more. We get rid of Tfidfvectorizer to use a simpler CountVectorizer. This solves the flaw mentioned above and now the user who appeared as the 6th matching user ranks 3rd.

Remark

Randomly exploring matches revealed something striking: some users with more shared interests were still ranked below some others with less shared interests.
Adding the percentage of shared interests out of all the interests of a given match led us to realize that the algorithm values whether the shared interests represent a large fraction of all the user's interest or not. For example, if A and B have an equal number of shared interest with the queried user but those represent respectively 48% of A interests and 53% of B interests. B will rank higher.

Adjusting the vectorizer’s parameters

As provided by scikit-learn's documentation the max_df parameter allows to set a threshold so that when the vocabulary is built, the vectorizer will ignore terms that have a document frequency strictly higher than the given threshold. In our case setting a low number relative to our dataset (1000) as threshold has the effect of discarding the 281 most common interests. Thus, our algorithm now seems to return people matching users based on their most peculiar and uncommon interests... which is something potentially interesting. However, this raises the question: are there interests so commonly shared that they can be ignored?
We take a stance of saying no. Hence we do not set a threshold for max_df.

Using a different distance metric

We duplicated the function to compute similarity and replaced cosine similarity with euclidean_distances from scikit-learn in order to compute user similarity and get the most similar vectors for a given one.
Nevertheless, the results were not as satisfying as those given by cosine similarity. For a specific user, using Euclidean distance returns users with less shared interests.

IV. Results

Model Evaluation and Validation

From the section above, we can argue that the final model is the result of a lot of refinement. Choosing CountVectorizer seems to be the right choice given the input data. TF-IDF is made for extracting what a document is about, ignoring the words that are too frequent. However, from our input data, the "bag of words" for each user are just the list of their interests, all of which important.

Let's explore and discuss the different parameters of our model:

input : We pass the column containing users' interests as input
encoding : our data is already encoded with the default utf-8 encoding.
decode_error : our data doesn't contain characters not of the given encoding
strip_accents : unnecessary
analyzer : we want each interest from Meetup data, as defined by one that is uniquely identified by an ID, to be a feature. Hence this parameter is unnecessary.
preprocessor : There's no need to override the default preprocessor.
tokenizer : We overrode the default tokenizer in order to preserve the orginal interests that can be made of several words!
ngram_range : There's no need to define a range for n-grams
stop_words : no stop words are present in our corpus
lowercase : 'False" is used to keep interests as they are (capitalized)
token_pattern : The token pattern is defined by our token pattern
max_df & **min_df** , **max_features** : As explained above, we consider that all interests should be taken into account regardless of their frequency.
vocabulary : the vocabulary is determined from the input documents, namely the users' interests
binary : This parameter which by default is set to False, was tried with True, setting all non-zero counts to 1. But the output of our algorithm was exactly the same.
dtype : type, optional

As expected our solution model returns a list of users ranked by similarity of interests with the queried user, with a similarity score between 0 and 1. Here's an example of output (with user #246) from our algorithm:

```
Selected user's interests: [u'Consciousness', u'Jewish', u'Spirituality',
u'Alternative Medicine', u'Meditation', u'Bilingual Spanish/English', u'Ph
ilosophy', u'Nature Walks', u'Outdoors', u'Animal Welfare', u'Dog Rescue',
u'Pug', u'Small Breed Dogs', u'Art', u'professional-networking']

Queried user has 27 interests

Match #1
Match score with 41839: 0.580258853186
Number of shared interests: 10 (91% of all 41839's interests)
Match #2
Match score with 65608: 0.544331053952
Number of shared interests: 8 (100% of all 65608's interests)
Match #3
Match score with 45050: 0.544331053952
Number of shared interests: 8 (100% of all 45050's interests)
Match #4
Match score with 70043: 0.544331053952
Number of shared interests: 8 (100% of all 70043's interests)
Match #5
Match score with 28156: 0.533760512684
Number of shared interests: 10 (77% of all 28156's interests)
```

The question is: how can we trust these results? How can we be sure that there's not another user in our data set that has a way more shared interests with our selected user(#246)? One way to validate that is to create a completely new and fake user, very similar to user #246 by taking the list of #246 interests and remove 3 from the list. We then run our algorithm and see if user #246 appears in the list of matches. The results are:

```
[ (246, 0.89442719099991586),
  (37528, 0.54554472558998102),
  (43095, 0.51639777949432231),
  (44523, 0.50000000000000011),
  (37634, 0.50000000000000011),
```

As expected User #246 ranks #1 in the list with a very high score of 89%. ☺

Justification

In order to compare the solution model with our benchmark we use the following metric. For a queried user i, we loop through the list of matches and compute the following for each match j:

Number of shared interest between i and j / Number of interests of i

We then look at the average in order to determine if the list contains a high number of people sharing a lot of interests with the queried user; as well as standard deviation.

Here are some results obtained by comparing our solution model with the benchmark model:

- **Solution model:**

From running several times our algorithm with samples of 1000 users, on average the top 10 recommended users share about 52% of the queried user's interests, with an average of standard deviations equal to 9%.

On average the number 1 recommended user shares about 56% of the queried user's interests

- **Benchmark model (gensim):**

On average the top 10 recommended users share about 55% of the queried user's interests, with an average of standard deviations equal to 9%.

The benchmark model seems to perform a bit better than our solution model in terms of average shared interests.

However, something important to consider is processing time: The benchmark model is able to execute 10 iterations (with a number 10 matching users returned) in 257 seconds as opposed to 0.76 seconds for our solution model! Our implementation of gensim can most likely be improved to make it faster or maybe gensim doesn't inherently use sparse matrices which makes it slower. As it is now, the benchmark's speed largely favors our solution model which clearly meet the requirements for solving the problem we defined.

V. Conclusion

Free-Form Visualization

The following shows the result of clustering after performing LSA with 500 components. It's awesome to see that after unveiling the top interests per cluster we have in clusters like #601 very related interests grouped together.

	Cluster 1	Cluster 201	Cluster 401	Cluster 601
Top terms per cluster:	Self-Improvement Business Strategy Live Music Meditation Art Professional Development Startup Businesses Dining Out Eating, Drinking, Talking, Laughing, Etc Exercise	Meditation Spirituality Energy Healing Spiritualism Yoga Alternative Medicine Reiki Holistic Health Fitness Travel	Online Marketing SEO (Search Engine Optimization) Startup Businesses Professional Networking Technology Marketing Fitness Travel Outdoors Investing	Italian Language Italian Culture Italiano Italian Food Travel Expat Italian Dining Out Italian Travel Wine Happy Hour

Reflection

In this project, we used data coming from Meetup's API that represent profiles of 82,186 members. Each member having indicated interests on their profile, we used this piece of information and treated each member's interests as a document or 'bag of words", and the whole set of documents as a corpus. In our case our corpus is our dictionary made of 21,892 unique interests.

It was very interesting to explore the distribution of these interests. But the core of the problem we attempted to solve was to recommend users to other users based on the similarity of their interests. So we turned our corpus of documents into a m x n matrix where m is the number of users and n the number of unique interests thanks to a CountVectorizer. We then used this matrix and computed the cosine similarity between the row of user with all the other rows and ranked them by similarity. This turned out to be the best solution after trying other vectorizers and similarity metrics.

An interesting analysis was also made using LSA to reduce the vector space from 21,892 to only 500 while keeping 85% of the variance explained. And performing clustering on top of that using K-means.

This project was challenging: from lists and data frames manipulation, to matrix computation and scikit-learn implementation. Even though the core of the solution is pretty simple, it helped me learn a lot about semantic analysis and machine learning.

Among the issues encountered, surprisingly, was memory usage and computational power, getting MemoryError or having my computer completely frozen happened a few times.

The final solution model seems satisfying as I stay fascinated by how powerful sometimes machines can be. Our final model is able to return the 10 persons with whom a given user shares the most interested among a list of +82,000 others in less than a second! A task so daunting that no human could undertake, let alone in such time. I am looking forward to implement it in an app.

Improvement

The algorithm provides a satisfying solution but I would like to continue analyzing the results that it returns, comparing them based on my own appreciation of similarity between users in order to potentially find ways to improve the solution. If a better solution exists I'm eager to implement it.

Else, the improvements that I'd like to make are related to the deployment of the algorithm to be used in a more user-friendly way:

- Turning the algorithm into an API that is consumable through a RESTful service.
- Hosting the source code on the cloud to make the API easily available.
- Being more flexible regarding the input data. Currently the solution only used data from the Meetup API that was turned into a data frame. It would be great to accept and mix different sources of data.
- Letting users create empty profiles instead of only pulling them from Meetup. Potentially users could start with no interests and indicate them to the API themselves. This raises the issue of data validation and consolidation. If one user enters "hiking" and another enters "Hiking", I would have to make sure that the algorithm treats those two as being the same interest, an issue that didn't arise from using Meetup API's data.