**Analyzing and Categorizing Instagram users throughout NLP**

Beltran Valle, Joaquín Casasús, Carlos Tapia and Manuel San Román

Ie School of science

P.º de la Castellana, 259, 28029 Madrid

E-mail: bvalle.ieu2020@student.ie.edu

Sentiment analysis tailored to a particular Brand -> Text classification.

(NER): Named entity recognition.

Instagram (LDA)

**Abstract**

Analyzing and categorizing influencers is an essential task to generate useful and handy recommendations to companies for their marketing campaigns. Different approaches include either content-based categorization approaches; in which influencers are categorized based on their content, or audience-based approaches in which demographics and interests of followers are used to categorize them. Brand collaboration approaches are also widely used to categorize influencers. In this paper we propose a hybrid approach combining both content-based categorization and audience-based approaches to generate powerful classifications that can be used to generate recommendations to companies interested in acquiring influencers for their campaigns. We will be using NER (Named entity recognition), LDA (Latent Dirichlet Allocation) and sentiment analysis techniques to identify useful patterns in influencers data.

NLP holds the title for being one of the technologies that is most rapidly evolving within the context of Artificial intelligence. It is no surprise, that communicating with computers has changed and enhanced almost every industry that has implemented this technology; like in the case of healthcare (Clinical documentation) or finance (Risk assessments), many different solutions can be found using NLP technologies that apply to very different industries. Both CNNs and RNNs allow for stronger sentiment analysis techniques. In addition, increasing availability of data gives room to further innovation. In this paper we will review different techniques of NLP and compare them in the context of recommending an influencer to company.

**Data gathering**

Regarding the data we produced a dataset with (number) rows and (number) columns. For this task we needed to use bot automation with Selenium. First, we scrapped thousands of different names from different users. We have optimized the algorithm so it works for any different Instagram account it may encounter during this process. In selenium, for instance, you can search By.XPATH or By.CSS, this allows the bot to search for clickable tags within the html structure of the Instagram webpage. Private accounts for example have a different structure, so the original algorithm must be modified for the bot to click where it is meant to click in those precise cases, where mis clicks may throw an error or directly lead you to another instance of the webpage where the structure might be completely different. Following this approach we scrapped data from their followers, likes, posts descriptions, user descriptions, post locations. In this research paper we will focus on the user and post descriptions at first hand. We have noticed a problem while gathering the data; Instagram users do not usually put the location of the post they upload. In fact, it is likely that information like this appear in the user description or in the post descriptions as well. Not only information about the content and audience of influencers can be found, collaborations with other brands, or even collaborations between influencers are more likely to appear here. There exist many posts where information about likes do not appear in the description, instead a text including brief information about the people who liked the post. This information could be used to get information for possible brand collaborations and even could be used to further confirm the contents the influencers frequent. We scrapped the first ten posts of every Instagram user; the text information will be stored and preprocessed. Then we took the average between these 10 posts to reduce them to one row for every influencer. In this case, I only need information about the user description, post descriptions and location, therefore I will create a column where all the descriptions will be gathered with a space separator. Sentiment analysis will be performed based on this column. To finalize with the preparations, we went through the process of lemmatizing and stemming. Lemmatizing and stemming are a must in the process. We return word back to their stem is easier to find them because the dictionary may not match the form of the word present in the text however, we know the stem is present. We do not need to use both, but in this case, we found more matches (I explain this in the next section) when introducing lemmatization as well therefore we will stay with both.

**Sentiment analysis**

You may think it is strange for us to use sentiment analysis in our model if we are only analyzing influencers data (not the comment received per post), however there are a few reasons why we followed this approach. Sentiment analysis will help us draw interesting conclusions form the data, like what kind of content the influencer followers is receiving, or maybe we want to engage brand collaboration, then sentiment analysis would be useful to understand the person values and know beforehand if that influencer will be happy and willing to participate in a certain campaign. The same can be said for negative feedback, if an influencer is constantly receiving negative feedback, we can analyze the main ideas and the words used by the influencer to understand this negative feedback and act. At the end we have defined a set of emotions including anger, trust, anticipation, disgust, fear, joy, negative, positive, sadness, surprise. The NRC-Emotion-Lexicon-Wordlevel-v0.92, includes these set of emotions and a dictionary of English words from a to z have these emotions related per word. For each row, you have a word the emotion and a third value which is either 0 if the emotion is not related with the word, or 1 if the emotion has a relation to the word.

Graphical user interface, text, application

Description automatically generated

Figure 1: Example of the NRC-emotion-lexicon with the word aback

With this file we just need to map the emotion to the word and check the value present in the NRC-Emotion-Lexicon. If it is one, we add one to the column where the row contains the specific word equal to the emotion we are targeting. In the end if we add up all the words, we get the total sum for each emotion.

Graphical user interface, application, background pattern

Description automatically generated

Figure 2: Result gathered from sample of data.

As we can see in the image above the first user has loads of matches between words and emotions. This is likely because the influencers use many adjectives or has very long descriptions. Therefore, we are not supposed to assume this specific influencer would be the best for any campaign, but we will need to look at their highest values and assume he/she would be the best for the companies that look for an influencer who transmits these principles. To get a more interesting result we decided to normalize the data. I think the most sensible metric we could use here is the z score to normalize the data. This is because like this we can get the overall sentiment the influencer transmits in their post in comparison to others. Below you can see a sample of the results.

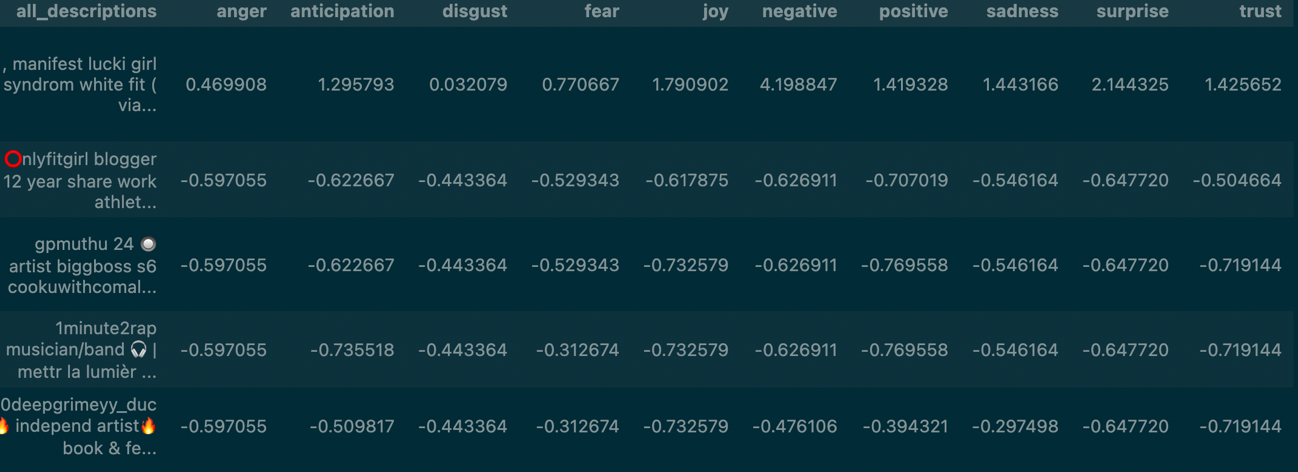


Figure 3: Normalized z scores for emotions.

As you can see from the results the first influencer would show their messages are overall negative in comparison to the rest of the dataset. This is interesting having in mind the highest value when doing the sums was for the positive values. With the results obtained from the text classification we could ask for a company to add their data to the database and include the kind of emotions they would like to transmit to their audience. Matches between the companies and the influencers can be drown from this analysis.

**Named Entity recognition.**

After thinking about how to get the greatest detail on the information present in the text, we came up with different ideas, however with NER we found out we could address many different issues. One of the main problems with our data is we have almost no information relating influencer and locations. This is because most of the people do not bother or don’t want to include their location in their posts. NER provides a solution easy to implement that can be super informative. With NER you can find recognizable names within text. This allows us to find organization names, cities, famous buildings… which can give us information on what places does this person like to visit, what influencers does he know, what brands does he/she like. To cope with our own problem, we thought this would be the most interesting solution. The next problem comes to mind when trying to find a suitable language detector for your model. It is widely known that this kind of algorithms are more efficient when given large corpora of text, therefore, we had to compare many different algorithms like cld3, langID or langdetect and choose the one that returned the greatest number of matches for places within our text. Bear in mind we will need this algorithm for two reasons; we need to know the languages present within our text, and secondly, we need to know what language we are reading to apply the correct pretrained model. The first step is to print a list with the languages present in your dataset. In this case, I will be using the spacy library where all the pretrained models for many different languages can be found in their webpage. You can either install them manually and add them to you model by including their path, or pip install them in your environment. In the end, we have decided to use langDetect even though before comparing them I thought the best one would be lingua; however, we got the best results with LangDetect our data. With others like langID, results were terrible, half of the English speakers from the data where being analyzed as Latin speakers. This is why I recommend trying the greatest number of models for your own data!

Once set, with all the libraries installed we start by detecting the unique different languages we have in our data. For this we can use the classify method for the langDetect library.

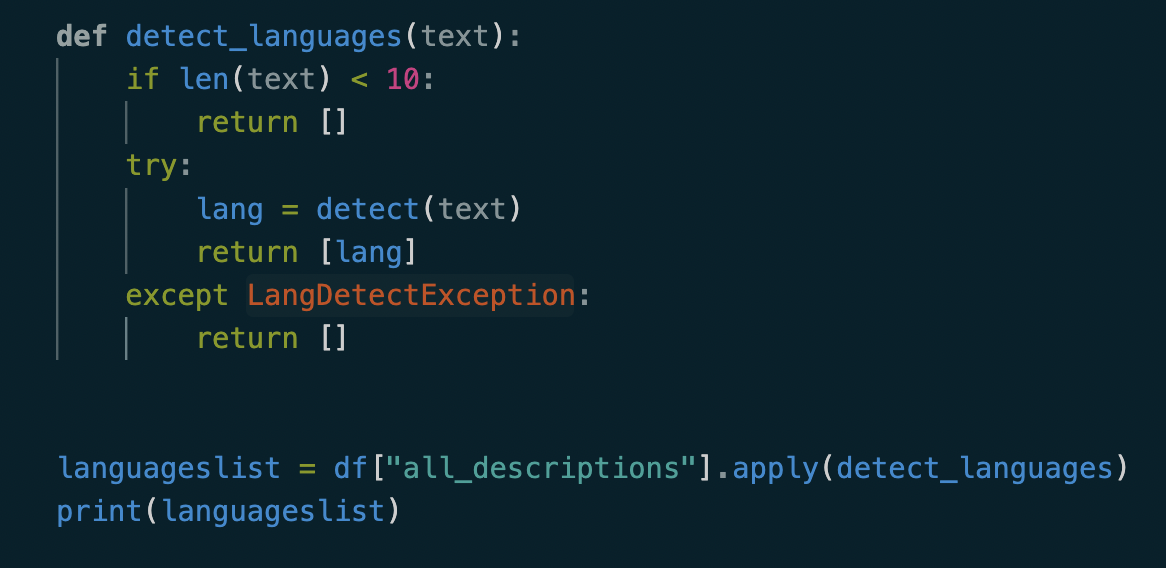


Figure 4: Example of langDetect.

This code will return a list of lists with all the texts having their language detected. The way this models work is by dividing the text in certain number of n-grams. For example, the brown fox down the street, was eating food. N=2 n-grams could be, the brown, brown fox, fox down, down the… This n-grams representation of the text can be later used to create vector-representations of the frequency with which each n-gram appear in the text. The more combinations of n-grams the greater the computational expense of the task. More powerful models like lingua use combinations from 1 to 5 n-grams providing more powerful models.

The code above returns a list with all the codes for the different languages present in the text. Now we can create a set to display all the unique values to identify the languages we will need to detect. The next step was to get the locations form text. For this we needed a few things. First, we need a nlp model capable of reading and understanding the language. With this model now we can understand Spanish text and give context to words with n-grams. Stop can mean various things; however, stop sign is clearly one thing. It is also necessary because there are some languages in which countries are traduced or spelled differently. For this reason we will need to use spacy.load() to load all the models for the languages present in our text. We also know that there are not pretrained models in the spacy library for every language present in our text, so we decided the best approach we could take was to get the frequency of the appearance of each language in the text. If there is a model available in Spacy that could complete the task we will use it, otherwise we will check its appearance in the text. If the appearance is too low, we will discard it, so we do not have to take it into account later.

Then, we load the models for every language present in our text, these models generally have this structure; en\_core\_web\_sm, for English, es\_core\_news\_sm, for Spanish. In the files, the first element is the language code; en for English and es for Spanish, while the last element sm stands for small. You can find md; medium or lg; large. Files ending in sm are faster but less precise in general, which makes them ideal for testing, while lg files have greater models with a greater number of parameters resulting in a better approach for the final prediction in general. After checking the frecuency for the appearance of each language in the data, we decided to load models for English, Arabic, Spanish, Italian, Portuguese, and French (all files ending in lg). As there exist no models for certain languages like in the case of Arabic we loaded an extra model, the xx\_ent\_wiki\_sm which includes nlp entity recognition for up to 50 unspecified languages. Thanks to this solution, we could cover many more languages we than what we initially expected to cover. Once loaded we had two options. Either we used the ent\_label and search by tag GPE, which stands for Geo-Political-Entity, or we could also use another library to retrieve the names form text. After a bit of research and after we had seen results searching by GPE, we decided we needed a library for the task. With the library location tagger, we could easily return the names of countries and cities found in the present text. I leave you an example of the implementation below.

Text

Description automatically generated

Figure 5: Example implementation of location tagger.

The results are stored to a new column called locations full. In the image below you can find an example of how the column looks.

Text

Description automatically generated

Figure 6: results detecting places. Look at the column “locationsfull”.

As we can see in the screenshot above the places found in text are retrieved and stored in a new column. In this case we can gather further information about places the person likes and visits and has helped us coping with the missing data we had in the location’s column. This second approach has allowed us to exploit our own data and refactor it into a richer data than it was before.

**LDA (latent dirichlet allocation)**

The last tool we will use will allow us to divide influencers by meaningful words in text. With LDA we can divide influencers into several topics by getting mixtures of words in different documents. It does this by analyzing the number of times these bags of words appear together in different documents. This is useful for our case since it will help us divide influencers for what they do. First, we used the WordCloud library. This library enables you to visualize the words in your data by the frequency with which they appear.



Figure : WordCloud distribution.

In this way we can see the words with highest appearance are stopwords like de la or que (Spanish.stopwords). This suggests that we should get rid of the stopwords of the languages with highest appearance within the text. For this purpose, I decided discard Spanish, English, French, Italian, Russian, Arabic, Indonesian and Portuguese stopwords. Now we had already preprocessed the text, so it won’t be necessary for our final task.

A screenshot of a computer

Description automatically generated

Figure 8:LDA example.

As explained before LDA uses bags of words that frequently appear together to show underlying topics we wouldn’t be able to see other way. In the corpora.Dictionary() file, we can find each unique word mapped to a unique Id. This allows us to draw numerical inputs from the data to later use the in LDA. The corpus.mm file contains a sparse matrix representation of these unique words mapped to unique ids. Both together are used to draw mappings between words and create bags of words that frequently appear together in different documents.

A screen shot of a computer

Description automatically generated with low confidence

Figure 9: Topic division LDA (6 topics)

This first result with 6 topics can help us understand further our data. In this case, we can easily point out a few problems and these can be caused by the skewness of our data. As we can see the 6 topics present have word like love, make, happi… suggesting the data in general is based on the same topic. This is possible and it happens in some cases in which there are too many influencers related with a similar topic; in this case aesthetics, style, fashion… There are a few things we should try here, and a quick implementation of k-means for text classification with elbow method to select the optimal number of clusters can prove useful to understand the different topics present in our data as well. In the case of k-means we received 6 as the optimal value of k after checking it with the elbow method. It is also extremally useful to print more and more words from each topic and see what the word with least weights represents and if they relate to more specific topics. Look at the results above and see how link is one of the words that appear the highest, this is just because there are many influencers who may leave their link below their post and they also specify it, however this does not mean they belong to a certain category. The word musician may have a much smaller weight because it appears a lot less in the data, but it is way more informative for us than link. The idea is to adjust the number of topics until you can clearly see different distributions of word that relate to different topics. If you never seem to find a useful distribution of topics you may try increasing the size of your data.