Success Prediction of Instagram Posts

Natural Language Processing

2021-2022

Authors

Sandeep Kumar Kushwaha

Jyoti Yadav

Zarmina Ursino

Source code available at : https://github.com/Zarmina97/NLP-project-

Summary

Abstract

Over one billion people use Instagram every month, which makes it one of the most popular social networks worldwide. Currently, there is an enormous scope market with the potential to be optimized to increase Instagram posts popularity and engagement. In this project, the main goal is to predict the number of likes given a post, using Machine Learning Models like LighGBM, XGBoost, Random Forest, SVM, MLP where we experimented with different types of features such as Meta features, NLP with different word representation & CV features. This is a use case that could be interesting in a real-life scenario, for influencers that would like to maximise their impact on people.

Introduction

Instagram is a photo-sharing platform that was launched in 2010. It has gradually gained a leading role among photo-sharing platforms, introducing several innovative features over times, including filters, stories, and an internal messaging system. These features have attracted not only ordinary users and photography enthusiasts, but also companies, organizations, and global brands, thanks to the possibility of exploring new business models and marketing strategies.

On the other hand, influencers having millions of followers can build communities around topics and niches. They use Instagram, as a visual platform to make regular posts and generate large followings of engaged people who pay close attention to their posts. For that reason, there is a special relationship between influencers and brands. Influencers know their followers and are conscious about what kind of content they want to consume, that way, it can help brands to communicate their messages. Commonly, Instagram influencers get paid by brands to make the promotion of a product and service. Currently, there is an enormous scope market with the potential to be optimized to increase Instagram posts popularity and engagement. So, for that reason we decided to predict the number of likes given a post. Using that system, it will generate an estimation so that posts can be optimized to gather the most amount of visibility as well as engagement.
In our project we are going to predict the number of likes given a post by building machine learning models, using meta-features, Natural Language Processing features & CV features. We can scan our work in two macro steps in Fig. 1:

1. The Feature Engineering Stage and
2. Supervised Learning Step

The first step includes the data pre-processing, thus we enriched the dataset with some derived information, as well as removing data whose contribution was negligible or not interesting for the class prediction purposes. In the second step for Supervised Learning Step that includes the classification model training and the success prediction of new posts. Later, in the Fig. 2 we describe the combination of models/features that we used and study each one's performance.

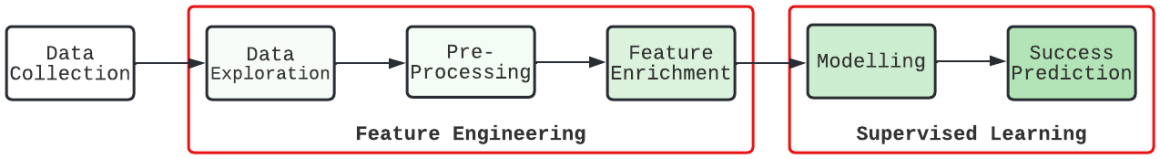


Figure 1 - Overall Workflow

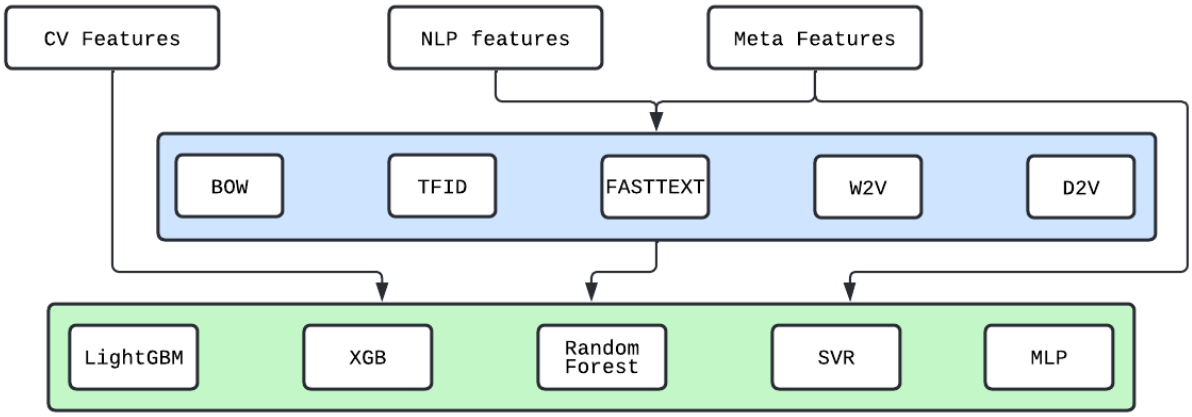


Figure 2 - Project Structure

Datasets preparation

We obtained the Instagram post and their details using a scraping tool, provided by the host company. The dataset was collected for the past 1 year i.e., from 1 Jan 2021 to 12 March 2022. The dataset consists of Instagram post & its details of 981 unique business users residing in UK. We got the following files to start the exploration and pre-processing part:

- ***rawdata.csv***: Contains all information about caption, biography, following, likes, posts_count, followers, datetime, comments etc. Its shape is (88516,37).
- ***industry.csv***: which contains information about Industry, Sub-category & Region. Its shape is (834,4).

We merged both the dataset mentioned above based on account name that is unique for every user.

CV features extraction

Apart from the textual/raw information obtained from the Instagram posts, we decided to use the image features as add-on. For that, we used the Image Analysis client Library of Azure. The Analyse Image service provided us with AI algorithms for processing images and returning information on their visual features. We chose the API services to extract following information: content tags & colour scheme detection. The colour values were converted into RGB format from text and later used as numeric feature for the next steps.

- ***data.csv***: this file containing content tags and colour scheme and is merged with *rawdata.csv & industry.csv* based on account name that is unique for every user.

Data Exploration through Clustering

At first, we decided to do some exploration for industry & subcategories. To reduce the number of features in our data set we deployed PCA (Principal Component Analysis) which tries to find the best possible subspace. It transforms our initial features into so-called components. These components are basically new variables, derived from the original ones, and they are usually displayed in order of importance. At the end of the PCA analysis, we aim to choose only two components, while preserving as much of the original information as possible. We incorporate the newly obtained PCA scores in the K-means algorithm. In this manner we can perform segmentation based on principal components scores instead of the original features. We need to determine the number of clusters in a K-means algorithm and we do that by calling the 'unique()' method on SUB-CATEGORIES feature. We add the names of the segments to the labels, and we map the clusters inside a new column for the mapping. To visualize our clusters on a 2D visualization we choose the two components and use them as axes with the help of matplotlib and seaborn library. Thanks to PCA we are sure that the first two components explain more variance than the others.

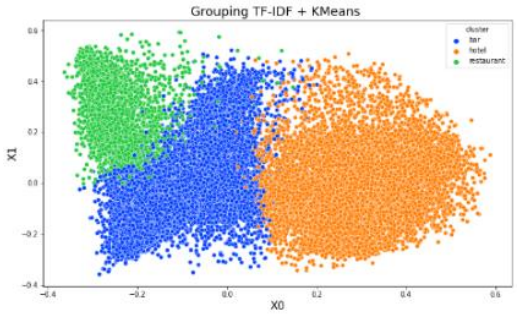


Figure 3 - Hospitality

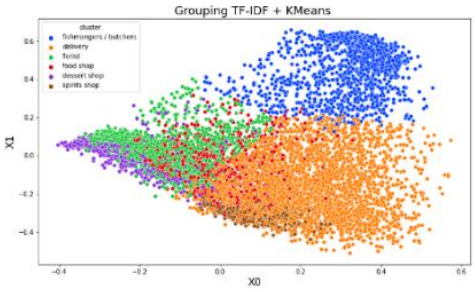


Figure 4 - Retail

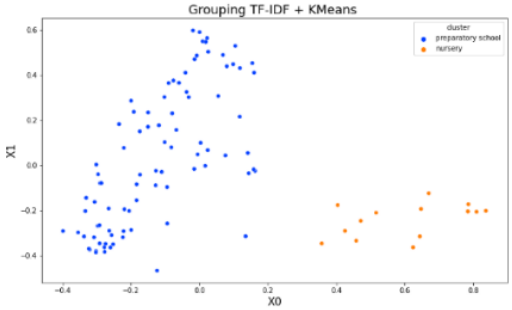


Figure 5 - Childcare

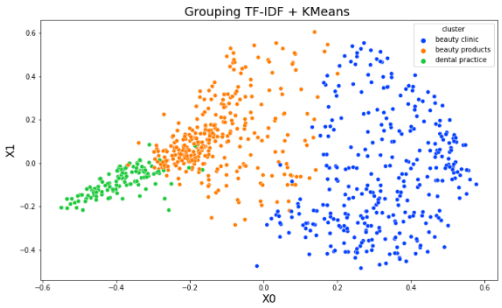


Figure 6 - Cosmetics

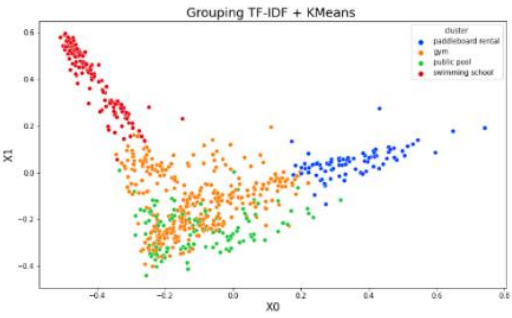


Figure 7 – fit sports

In the Fig. 3 & 4 we can observe most of the posts in UK belongs to the industry of hospitality, which has the categories 'bar', 'restaurant', 'hotel' and retail, which has the categories 'dessert shop', 'florist', 'spirits shop', 'fishmongers/butchers' and 'delivery'. Similar exploration has been performed in multiple domains in Fig. 5, Fig. 6, and Fig. 7.

Data pre-processing

The pre-processing functions are defined which are useful for performing pre-processing on the different datasets. The functions implemented are:

- *lower():* the textual data is converted to lowercase.
- *clean_mention():* removes the mentions of the users from the caption.
- *clean_hashtags():* removes the hashtags from the caption.
- *replace_special_characters():* it replaces special characters, such as parenthesis, with spacing character
- *replace_br()*: it replaces br characters
- *filter_out_uncommon_symbols():* it removes any special character that is not in the good symbols list (check regular expression)
- *remove_stopwords():* removes the stop words based on NLTK corpus.
- *strip_text():* it removes any left or right spacing (including carriage return) from text
- *remove_brackets_from_list():* it removes the [] brackets from the each of the list

In *utils.py* the functions implemented are listed below and they are necessary to build the models:

- *hyperparameterTuning_RandomForest():* It contains the hyperparameter tuning using Randomized search CV
- *hyperparameterTuning_XGBoost():* It contains the hyperparameter tuning using Grid search CV.

- *hyperparameterTuning_MLP():* It contains the hyperparameter tuning using Grid search CV.
- *plot_feature_importance():* It's a function to plot the feature importance for XgBoost & Random Forest.
- *featureScore():* It gives the scores to every feature.
- *folderPath():* The path of the current folder.
- *metrics():* It defines the metrics like RMSE, R2 score, RMSLE, MAE etc.
- *normalizing():* It normalizes the whole dataset.
- *removeColumnContainString():* It removes the irrelevant columns of type string.
- *Feature_Extraction():* It is a function to create the BOW & TF-IDF.
- *clean_dataset():* It removes any kind of NaN's or infinity values from dataset.

MetaFeatures

We created the base model with the meta-features. These characterizations, also called meta features, describe properties of the data which are predictive for the performance of machine learning algorithms trained on them. In *Metafeatures.ipynb* we consider the numerical features likes following, likes, post_count, followers, datetime etc. We generated some of the features from the raw features as described below:

Features	**Description**
caption_length	Length of the caption
biography_length	Length of biography
po_co	Post_count divided by comments
Pof	Post_count divided by following
user_count	Count of the number of posts made by a user
profile_name_len	Length of the name of user's profile
Fol	Followers divided by following
Act	Comments divided by followers
Pos	Post_count divided by followers
comments_max	Max comments of user
comments_min	Min comments of user
comments_mean	Mean comments of user
comments_std	Standard deviation of the mean comments of users
comments_followers	Multiplication of comments & followers
followers_comments_mean	Multiplication of comments_mean & followers
fol2	Multiplication of followers & their followers
fol_pos	Multiplication of fol & pos
fol_pow	Multiplication of followers & following
po_co_pow	Multiplication of posts_count & number of comments
comments_mean_diff	Comments divided by comments_mean
Dow	Days of the week (from Timestamp)
Hod	Hour of the Day (from Timestamp)
Date	Date (from Timestamp)

From clustering it was noticed most of the post belongs to hospitality clusters like hotels, bars and restaurants. For now, we decided to work only with the industry feature and hence we drop the columns sub-category. Then we make use of pandas 'get_dummies()' function to create dummy Industry variables from pandas objects in Python.

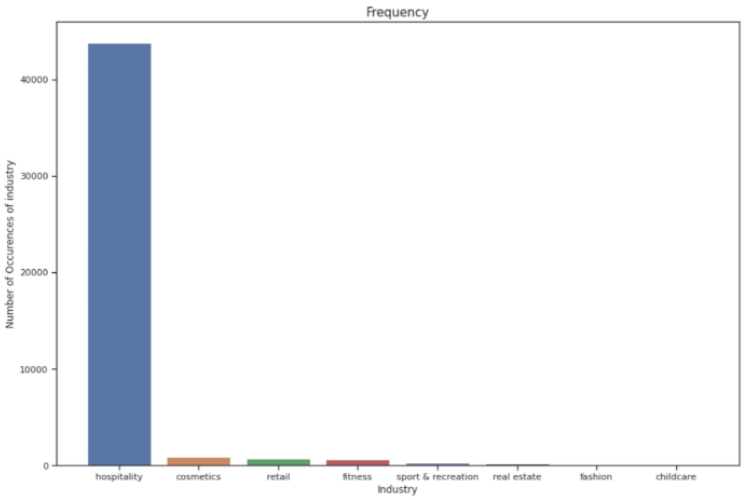


Figure 8 - Industry

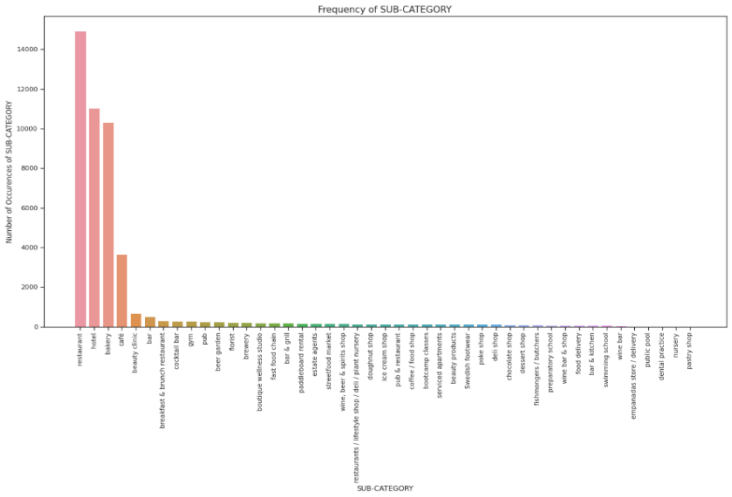


Figure 9 – Sub-Category

The datetime was initially in the form of timestamp. It was useful to generate new features from the timestamp like date, weekday, hour from the timestamp. Let's look at the Weekday feature where 0 - Monday, 1- Tuesday,2- Wednesday, etc. From the Fig. 10 it was observed that the frequency of post from the user started increasing from the Wednesday, Thursday and Friday and started to decrease during the weekend. Let's look at the frequency of post in the Fig. 11 on the hour of the day where it was observed that

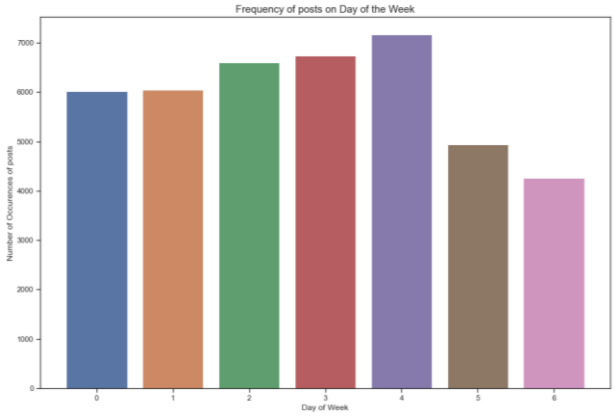


Figure 10 – Frequency of post on Day of the week

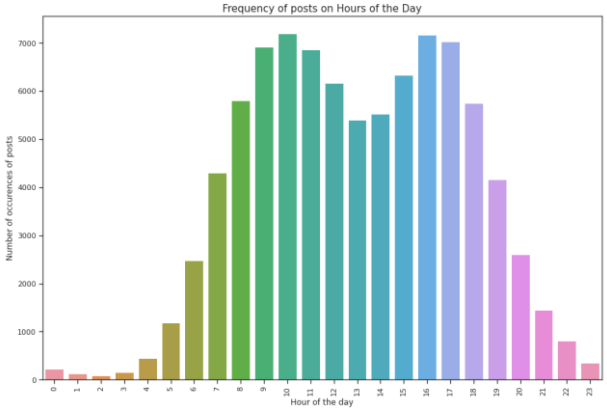


Figure 11 – Frequency of posts on Hour of the day

during 7 am to 10 am the users were active and then the frequency of post slightly decreases in the afternoon 11 am to 13 pm. It starts increasing again from 13:00 to 17:00. It starts decreasing a bit from 19:00 to 23:00.

In the Fig. 12 it was observed from the behaviour of the users that the greater number of followers they have the greater number of likes they get. Let's look at the behaviour of the users with respect to followers. It was observed that the users have their most of the activity at 16:00, 17:00 & 11:00 on Wednesday, Thursday & Friday.

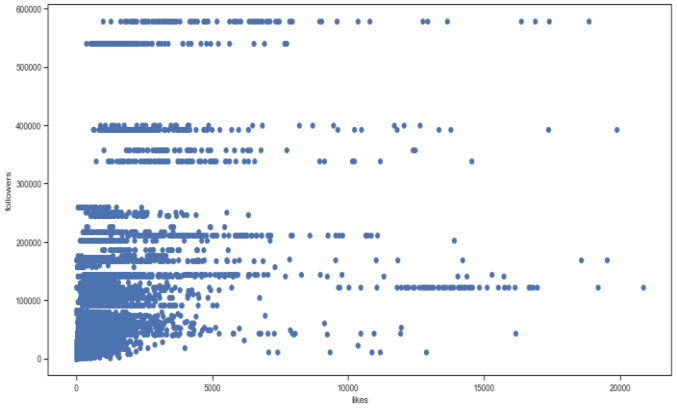


Figure 12 – Likes vs followers

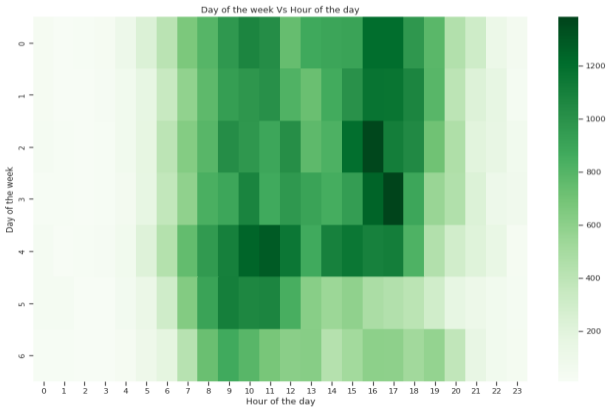


Figure 13 – Day of the week vs Hour of the day

We decided to remove other columns like 'INDUSTRY_fashion', 'INDUSTRY_retail', 'profile_name', 'REGION', 'biography', 'INDUSTRY_childcare', 'INDUSTRY_fitness', 'INDUSTRY_sport & recreation', 'INDUSTRY_childcare', 'INDUSTRY_real estate', caption. Let's look at the correlation plot in the Fig. 14 where we observed that there is a high correlation of likes with followers, comments_min, comments_max, followers_comments_min, fol_pow, following. We can draw inferences that likes on the post mainly depends on the users' comments, followers, & following etc.

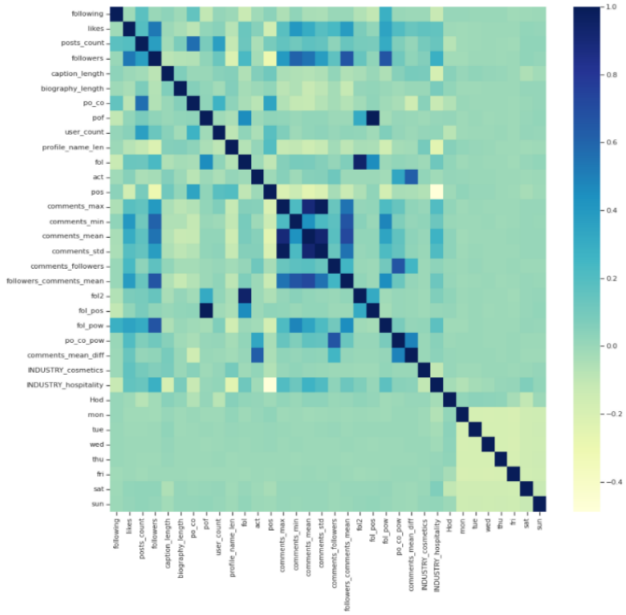


Figure 14 – Correlation plot

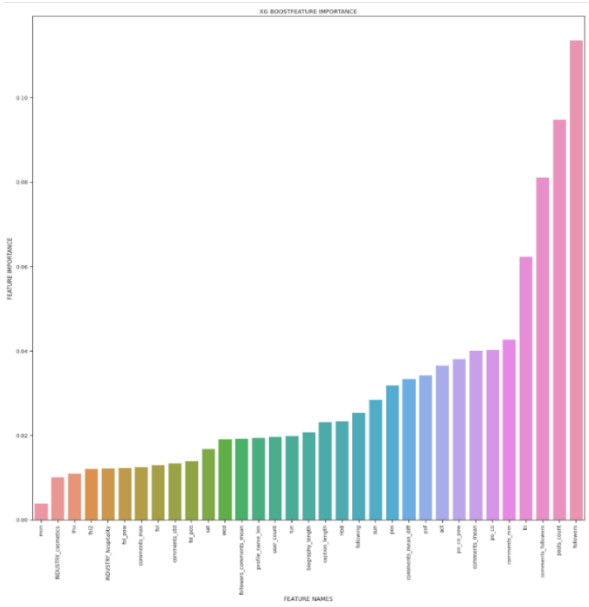


Figure 15 – Feature importance plot

From the Fig. 15 It can be inferred that the most important features are mon, INDUSTRY_cosmetics, thu, fol2, INDUSTRY_hospitality, fol_pow, comments_std, fol_pos, sat, wed, followers_comments_mean, profile_name_len, user_count, tue, biography_length, caption_length, Hod, following, sun, pos, comments_mean_diff, pof, act, po_co_pow, comments_mean, po_co, comments_min, fri, comments_followers, posts_count, followers.

NLP Features

In the NLP Features we make use of caption and the meta features for accurate prediction of the likes. The below features are used:

Features	**Description**
Caption	Caption of the post
caption_length	Length of the caption
Textblob_subjectivity	Opinion and judgements

Textblob_polarity	Sentimental of the post
Hashtags	Hashtags extracted from caption
Hashcounts	Count of Hashtags used in the caption
Mentions	Mentions extracted from caption
mention_count	Count of Mentions used in the caption
Emoji	Emoji extracted from caption
emoji_count	Count of Emojis used in the caption
emoji_text	Emoji symbols converted into text
hashtag_popularity	Hashtag popularity w.r.t to followers

At first, we decided to make use of sentiment as a feature. For that we make use of TextBlob analysis Library which reads the caption and provide us the sentimental subjectivity & sentimental polarity. Whenever we pass the sentence into Text blob it gives the output for polarity in the range of [-1,1] whereas the values towards the -1 refers to the more negative sentiment and the values towards +1 refers to the more positive sentiment. Similarly, subjectivity is the output that lies within [0,1] and refers to the personal opinions and judgements.

There are many other features within the caption that utilizes text including hashtags, emoji's, mentions. For the purpose of our project, we make use of advertools library which is used to extract entities from social media posts with the help of 'hashtags_summary()' which helps us to extract the hashtags like hashcounts, mentions, emoji ,mention_count, emoji_count and emoji_text. In the Fig. 16 it was observed that the top hashtags in the users' captions were #london, #workout, #eidenburg, #fitnessmotivation etc. In the Fig. 17 the top mentions used in the captions are @thegymgroup, @deliveroo, @kennethculhane, @thedysartpetersham, @crossfituk.

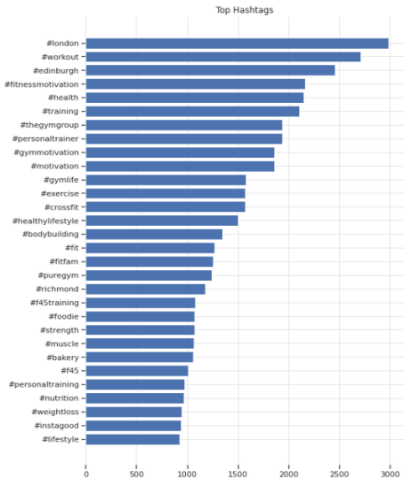


Figure 16 – Top Hashtags

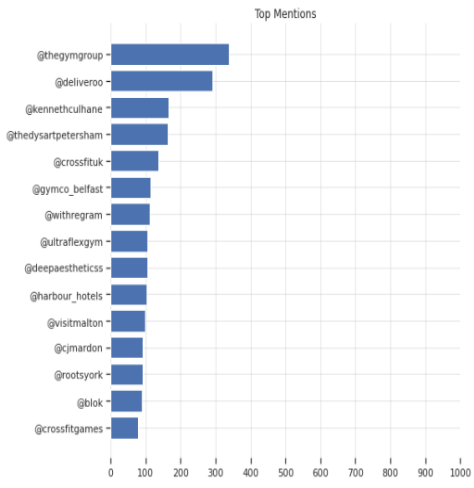


Figure 17 – Top Mentions

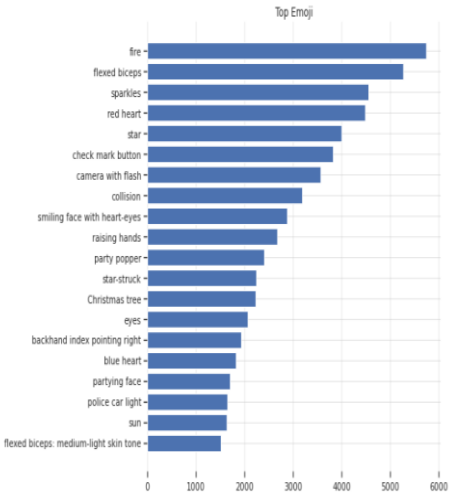


Figure 18 – Top Emoji's

In Fig. 18 it was observed that the top emojis are fire, flexed biceps, sparkles, red heart, star, check mark button. So far, we have counted the entities that we wanted and got absolute count of each in different formats and summaries.

We want to know how much impact these entities have on the engagement of the users. Let's suppose a person with 100 followers posted using the hashtag #workout in their Instagram post. Therefore, those post are expected to potentially 100 people. Let's also assume that another person with 20,000 followers used the hashtag #london in this post. Then which hashtag reached more users? Obviously #london would have better popularity because the people having more followers have more ability to make hashtags popular. We take the absolute frequencies of the hashtags and its corresponding weighted frequencies based on the follower's count. Then we calculate the relative value out of it by dividing the weighted frequency over absolute frequency. Using this, we calculate the hashtag popularity based on user followers (refer for more info Here).

Then we performed pre-processing as mentioned using the below pipeline. Let's explore the most common words after pre-processing used in the captions in the Fig. 19: Weekend, link, bio etc. In the Fig. 20 the hashtags mostly used by the users are fitness, gym, fitness motivation etc. In the Fig. 21 the mentions mostly used `by the users are @thegymgroup, @thedysartpetersham etc.

Figure 19 – Common words in caption

Figure 20 – Common words in hashtags

Figure 20 – Common words in Mentions

Words Representations

We decided to freeze above features & derived meta features for captions, hashtags and mentions using the below word representation techniques.

Bag Of Words – (BagOfWords.ipynb)

The bag-of-words (BOW) model is a representation that turns arbitrary text into fixed-length vectors by counting how many times each word appears. We splitted the words and used scikit-learn function 'CountVectorizer()' to vectorize the caption, hashtags, emoji's text & mentions. This resulted in a very sparse matrix. To reduce the sparsity of the matrix we employed a technique where we would only use the top words in the caption instead all the words. Thus, we were able to work with the top 100 words when evaluating our features. We created a new vocabulary of caption which are greater than 4 word-length. Then we combine it with the existing data frame.

TF-TFIDF - (TFIDF.ipynb)

TF-IDF stands for Term Frequency — Inverse Document Frequency and it is a statistic that aims to better define how important a word is for a document, while also considering the relation to other documents from the same corpus. We made use of scikit-learn Tf-idf function as 'TfidfVectorizer()' with only top 100 words. We created a new vocabulary with captions word-length >=4 then we combined these with the existing data frame.

Word2Vec

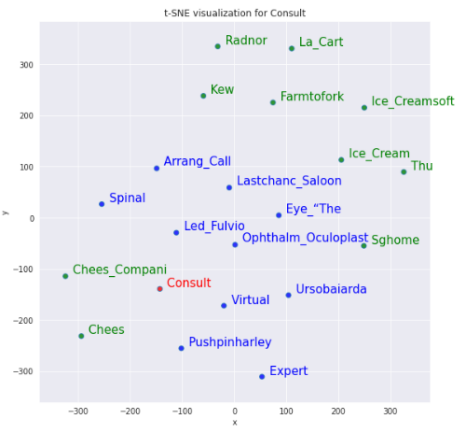


Figure 22 - Visualization of word2vec

Word2vec embeddings are modern way of representing words as vectors. Its objective is to redefine word features into low dimensional feature vectors by preserving the contextual similarity in the corpus. Word2Vec model takes a text corpus as input and produces the word embedding vectors as output. In Fig. 22 we plotted in seaborn the results from the t-SNE dimensionality reduction algorithm of the vectors of a query word, its list of most similar words, and a list of word: query word 'Consult' (in red), its similar words in the model (in blue) and other words from vocabulary (in green).

Word2Vec has two different techniques for implementation: CBOW and skipgram.

Word2vec CBOW – (word2vec_CBOW.ipynb)

It predicts the target word from the context. We first tokenize the caption, emoji Text, hashtags and mentions.

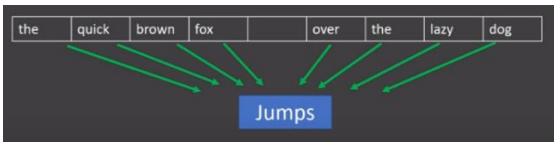
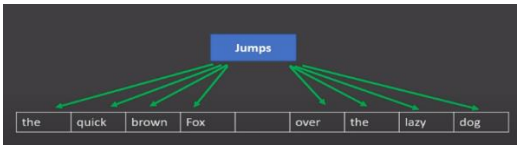


Figure 23 - word2vec CBOW

Parameter	Values	Description
Size	100	described number of features
Window	5	Context window size
Min_count	2	Ignores all words with total frequency lower than this.
Sg	0	Training algorithm: 0- CBOW.
Hs	0	If 1, hierarchical SoftMax will be used for model training. If 0, and negative is non-zero, negative sampling will be used.
Negative	10	If > 0, negative sampling will be used, the int for negative specifies how many "noise words" should be drawn (usually between 5-20). If set to 0, no negative sampling is used.
Workers	32	Number of worker threads to train the model
Epoch	20	Number of training epochs

Since our data contains captions and not just words, we'll have to figure out a way to use the word vectors from word2vec model to create vector representation for an entire caption. We decided to take the mean of all the word vectors present in the caption. The length of the resultant vector will be the same, i.e., 100. We will repeat the same process for all the captions in our data and obtain their vectors. Now we have 100 word2vec features for our data.

Word2vec - (Word2vec_skipgram.ipynb)

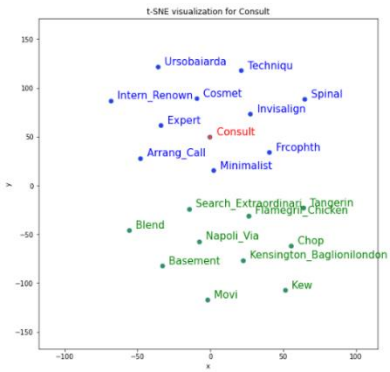


It predicts the context words from the target. Here also we use the above parameters except sg=1 for Skipgram Model. We can do so by using Gensim library.

Figure 24 - word2vec Skipgram

Doc2Vec – (doc2vec.ipynb)

Doc2Vec model is an unsupervised algorithm to generate vectors for sentences, paragraphs & documents. This approach is an extension of the word2vec. The major difference between the two is that doc2vec provides an additional context which is unique for every document in the corpus. This additional context is nothing but another feature vector for the whole document. This document vector is trained along with the word vectors. To implement doc2vec, we must provide labels or tag each tokenised caption with unique IDs. We can do so by using Gensim's 'LabeledSentence()' function.



As we did for Word2Vec, in Fig.23 we plot in Seaborn the results from the t-SNE dimensionality reduction algorithm of the vectors of a query word, its list of most similar words, and a list of word: query word 'Consult' (in red), its similar words in the model (in blue) and other words from vocabulary (in green).

Figure 25 – Visualization for doc2vec

Then taking the mean of the word vectors present in the caption, hashtags, emoji text & mentions.

Parameter	Values	Description
dm	1	Defines the training algorithm. If dm=1, distributed memory (PV-DM) is used. Otherwise, distributed bag of words (PV-DBOW) is employed
Vector_size	100	Dimensionality of the feature vectors.
Window	5	The maximum distance between the current and predicted word within a sentence.
Negative	7	If > 0, negative sampling will be used, the int for negative specifies how many "noise words" should be drawn (usually between 5-20). If set to 0, no negative sampling is used
Min_count	5	Ignores all words with total frequency lower than this.
Workers	32	Number of worker threads to train the model (=faster training with multicore machines).
Alpha	0.1	The initial learning rate
Epoch	20	Number of training epochs

FastText

FastText is an extension to Word2Vec proposed by Facebook in 2016. Instead of feeding individual words into the Neural Network, FastText breaks words into several n-grams (sub-words). For instance, the trigrams for the word *apple* is *app, ppl*, and *ple* (ignoring the starting and ending boundaries of words). The word embedding vector for *apple* will be the sum of all these n-grams. After training the Neural Network, we will have word embeddings for all the n-grams given the training dataset. Rare words can now be properly represented since it is highly likely that some of their n-grams also appears in other words.

```
modelFastTextSkipGramCaption.wv.most_similar("Gastroenteritis")
```

```
[('arthritis', 0.7127257585525513),
 ('osteoarthritis', 0.7120764851570129),
 ('gastrointestinal', 0.6904004216194153),
 ('colitis', 0.669417142868042),
 ('herniating', 0.6680669188499451),
 ('impulses', 0.6653196811676025),
 ('alstroemeria', 0.6634736061096191),
 ('gastropub', 0.6611387729644775),
 ('pingdemic', 0.6587487459182739),
 ('materialharvest', 0.6473618745803833)]
```

Figure 26 - FastText

Let's consider a word *Gastroenteritis*, which is rarely used and does not appear in the training dataset. Even though the word *Gastroenteritis* does not exist in the training dataset, it is still capable of figuring out this word is closely related to some medical terms. If we try this in the Word2Vec defined previously, it will pop out error because such word does not exist in the training dataset. Although it takes longer time to train a FastText model (number of n-grams > number of words), it performs better than Word2Vec and allows rare words to be represented appropriately.

It has two different approaches: CBOW and skipgram. These approaches are described above in Word2Vec section.

FastText – (FastText_Skipgram.ipynb)
The parameters we used same as for word2vec skip gram.

FastText – (FastText_CBOW.ipynb)
The parameters we used same as for word2vec CBOW.

CV features

We begin by connecting to the Azure blob in the file *data_collection.ipynb*, which contains all the images of the post etc in the *raw.csv* file. The images from the blob are accessed via SAS (shared access signature) using the function *get_blob_sas()*. The *get_cv_prop()* connects the SAS values with the *CvApiConnector()* and the computer vision API extracts the required information and converts all the features of a particular image into a list which is provided by the function *get_cv_prop()*. Since, the number of images is too large (85k+), we divided the images into batch-size of 3600 and performed the cv feature extraction, which are later combined into a single data frame: *dfComplete_cv.csv* using the *image_url* as common column.

Features	Description
Tags	Main contents in the images

Confidence score	Score with which tags were likely a match
Accent colour	Contains the main colors involved in the image
Is_bw	Information whether image is black & white
Dominant_colors	Most prominent colors in the image(usually 1 or 2)
Bg_colors	Background colors in the image
Fore_colors	Foreground colors in the image

Later we use the file *cv_features_preprocessing.ipynb* to perform pre-processing on the CV features. For columns containing textual information as string, we use regex to remove special characters and retain the words in a list. For columns that are supposed to be numerical, we perform regex to extract the numbers from the strings that are later converted into float values. The complete pre-processed data frame is saved as *data.csv*. We make an experiment of using metafeatures & NLP features using FastText skipgram Representation and use CV features to feed to the models mentioned below. We transform whole features into a given scale using Minmax Scaling. This estimator scales and translates each feature individually such that it is in the given range on the dataset, e.g., between zero and one. We decided to remove the NaNs and any kind of infinity values which is too large for data types before fitting the regression model using '*clean_dataset()*' function. We make use of sklearn function 'train_test_split()' and split the dataset into random train and testing subsets. The training set used is 80% and 20% of the dataset for evaluation with the *random_state* variable as 2022 to produce the fixed set of output across the multiple function calls.

Algorithms Overview

To model the data, we consider the five models discussed below.

LightGBM

The LGBM (Light Gradient Boosting Model) is used which has higher accuracy and a faster training speed with low memory utilization. It provides comparatively better accuracy than other boosting algorithms and handles overfitting much better while working with smaller datasets.

XGBoost

XGBoost (Extreme Gradient Boosting) is a scalable, distributed gradient-boosted decision tree (GBDT) machine learning library. It provides parallel tree boosting and is the leading machine learning library for regression, classification, and ranking problems.

Random Forest

A random forest is a supervised algorithm that uses an ensemble learning method consisting of a multitude of decision trees, the output of which is the consensus of the best answer to the problem. Random Forest can be used for classification or regression.

SVM

SVM is one of the supervised algorithms mostly used for classification & regression problems mainly for irregular distributed data. It provides a very useful technique within it known as kernel and by the application of associated kernel function we can solve any complex problem

MLP

MLPClassifier stands for Multi-layer Perceptron classifier which in the name itself connects to a Neural Network. Unlike other classification algorithms such as Support Vectors or Naive Bayes Classifier, MLPClassifier relies on an underlying Neural Network to perform the task of classification.

Best Parameters after Hyperparameter Tuning

Parameters Used	LGBM	XGB	RF	SVR	MLP
Activation					reLU
alpha					0.001

Column sample by tree	0.8	0.7			
dual				True	
Layer size					(50,50,50)
C				1.0	
loss				epsilon_insensitive	
epsilon				1.5	
Intercept scaling				1.0	
solver					Adam
Learning rate	0.01	0.03			constant
Max depth	8	5	10		
Min child weight	1	4			
Min split gain	0.0222				
Min Sample split			10		
Min sample leaf			4		
N estimators	35000	500	20		
Max Feature			sqrt		
Max iter				1000	
Tol				0.0001	
Num of leaves	966				
Reg alpha	0.04				
Reg lambda	0.073				
Subsample	0.6	0.7			
Number of threads		4			
Verbose				0	
Bootstrap			True		

We performed the hyperparameter tuning and found out that best parameters are the ones mentioned in the above table.

Error Analysis

The evaluation metrics used are RMSE (Root Mean Squared Error), R2 Score, MAE (Mean Absolute Error), RMSLE (Root Mean Squared Logarithmic Error) and Max Error.

Metrics	**Base Model**	**BOW**	**TF-IDF**	**Word2Vec Skip gram**	**Word2Vec CBOW**	**Fast Text Skip Gram**	**Fast Text CBOW**	**Doc2vec**	**CV**
LightGBM									
RMSE	0.008620	0.008644	0.008625	0.007498	0.007447	0.007131	0.007084	0.009629	0.008122
R2 Score	0.577815	0.582151	0.583974	0.610671	0.621513	0.605669	0.610846	0.522325	0.577527
MAE	0.001825	0.001836	0.001838	0.001773	0.001787	0.001858	0.001857	0.001820	0.001765
RMSLE	-4.75371	-4.75086	-4.75304	-4.893115	-4.899883	-4.94323	-4.94984	-4.64295	4.813200
MaxError	0.533904	0.537682	0.538299	0.323809	0.304083	0.197807	0.194891	0.864437	0.494410
XGBoost									
RMSE	0.007876	0.007805	0.008174	0.006736	0.007082	0.006502	0.006267	0.009472	0.007456
R2 Score	0.647521	0.659310	0.626353	0.685829	0.657789	0.672172	0.695508	0.537838	0.643937
MAE	0.001464	0.001491	0.001510	0.001435	0.001437	0.001472	0.001466	0.001472	0.001377
RMSLE	-4.84393	-4.85293	-4.80676	-5.000359	-4.950260	-5.03558	-5.07251	-4.65945	-4.89871
MaxError	0.520647	0.484182	0.526286	0.234300	0.293903	0.183032	0.174488	0.889369	0.479147
Random Forest									
RMSE	0.007606	0.008736	0.008484	0.007624	0.007576	0.006998	0.007174	0.009653	0.009934

R2 Score	0.671264	0.573200	0.597485	0.597488	0.608360	0.620265	0.600948	0.520011	0.368012
MAE	0.001413	0.001714	0.001640	0.001672	0.001704	0.001717	0.001761	0.001690	0.002498
RMSLE	-4.87880	-4.74026	-4.76955	-4.876465	-4.882802	-4.96209	-4.93728	-4.64053	-4.61182
MaxError	0.509198	0.578521	0.533026	0.389754	0.271130	0.204303	0.244427	0.866844	0.605138
SVR									
RMSE	0.013675	0.013762	0.013762	0.012425	0.012539	0.011823	0.011823	0.011841	0.012907
R2 Score	-0.06264	-0.05914	-0.05914	-0.069177	-0.072932	-0.08390	-0.08390	0.277668	-0.06690
MAE	0.003320	0.003252	0.003252	0.003161	0.003269	0.003290	0.003290	0.003542	0.003232
RMSLE	-4.29217	-4.28581	-4.28581	-4.388005	-4.378898	-4.43767	-4.43767	-4.43617	-4.35000
MaxError	0.639221	0.691846	0.691846	0.523607	0.387544	0.363963	0.363963	0.960571	0.639221
MLP									
RMSE	0.009841	0.010085	0.010117	0.009621	0.009720	0.008891	0.009380	0.011841	0.010449
R2 Score	0.449671	0.431302	0.427629	0.359014	0.355222	0.387016	0.317820	0.277668	0.300733
MAE	0.002749	0.002363	0.002454	0.003246	0.003136	0.003318	0.003789	0.003542	0.002813
RMSLE	-4.62117	-4.59674	-4.59352	-4.643823	-4.633520	-4.72266	-4.66918	-4.43617	-4.56124
MaxError	0.561765	0.639808	0.641612	0.484641	0.354472	0.290263	0.305460	0.960571	0.591780

From the above observation we conclude that the best performing word representations are Word2vec and FastText. The best performing model is XGBoost highlighted in the table with green colour. Considering the four metrics and taking into consideration the R2 score we found out that both word2vec and FastText performed well. But FastText has the highest R2 score in both the cases. We can conclude that NLP features has an important role to play as compared to the CV features. The worst performing model was doc2vec.

Future Improvements

- The R2 score can be improved further.
- It is worth trying to use a pretrained models like BERT.
- We can increase the dimensions of the embeddings for improved results.
- The number of comments could be the alternative variable to measure the popularity of a post if the number of likes are removed in the future as mentioned in the article below.

According to an interesting article (more info at Instagram removing likes), Instagram is testing to hide the number of likes on posts in several countries, including Australia and Japan. Likes counters will be hidden, so users can still see how many likes a post obtained but will be visible to only the author for the Instagram post. Adam Mosseri, Instagram's CEO, has said that removing likes was the platform's way to "depressurize" Instagram for young people. Data science teams at Facebook (which owns Instagram), believe that hiding likes might motivate people to post more frequently and it might extend the length of time that people spend on the app.

References

[1] https://docs.microsoft.com/it-it/azure/search/cognitive-search-concept-intro
[2] https://towardsdatascience.com/predicting-the-popularity-of-instagram-posts-deeb7dc27a8f
[3] https://github.com/shatha2014/FashionRec/tree/master/wordvecs
[4] How does FastText classifier work under the hood?
[5] Multi-Class Text Classification with Doc2Vec & Logistic Regression
[6] https://iajit.org/portal/PDF/Vol%2018,%20No.%201/19395.pdf
[7] https://upcommons.upc.edu/bitstream/handle/2117/339937/152579.pdf?sequence=1&isAllowed=y
[8] https://cjqian.github.io/docs/instagram_paper.pdf