SOCIAL MEDIA MANAGEMENT PROJECT

UNIVERSITY OF CATANIA

DEPARTMENT OF COMPUTER SCIENCE

Instagram Hashtags Generator

Author:
Rosario Scavo

Supervisor:
Prof. Antonino Furnari

August 18, 2020

Abstract

This project's main goal remains a "proof of concept" based on what I have studied through the material of the subject "Social Media Management".

Instagram allows its users to use hashtags, a maximum of 30 hashtags per post precisely. Hashtags are used to make a post easier to be discovered by other users. Some hashtags are more popular than others because are used independently of the context, however, some are correlated with each other(e.g., a picture of two friends could be correlated with the tags "people" and "friends")

The idea is to analyze some main tags and seek out the most common hashtags used with them, removing the ones that are known to be used to gain more likes and comments[1]. "Instagram Hashtag Generator" analyzes an image and gives back some hashtags that are based on a prediction of binary classifiers. It is possible to use two different classifiers, a classifier based on Logistic Regression and the other one on k-nearest neighbors.

Contents

1 Introduction

As "proof of concepts" for the subject "Social Media Management", I decided to create a program with an approach on a machine-learning algorithm, based on the concepts and libraries studied with the teacher. The program has been coded in python, considering that it is the language used during the lessons.
As explained above, the program has the aim to show to have understood be able to use the basic concepts behind the machine learning approaches. Therefore, the performance, the logic behind the data transformation, and the dataset's quality are secondary.

Instagram is the 5th most used social network globally, with more than 1082 millions of users [2]. On Instagram, it is possible to use **hashtags** to make a post more visible and searchable. The analysis of hashtags could lead to object recognition and even the context recognition of an image. For example, given an image with some people, analyzing the hashtags, seeking out that it is a picture of a party is possible (see Figure 1).

However, given an image as input, the goal is to produce a list of tags that the user could use, based on the dataset. Indeed, it should not be surprising if object recognition does not work adequately, considering that most of the pictures are out of the main tag's context.

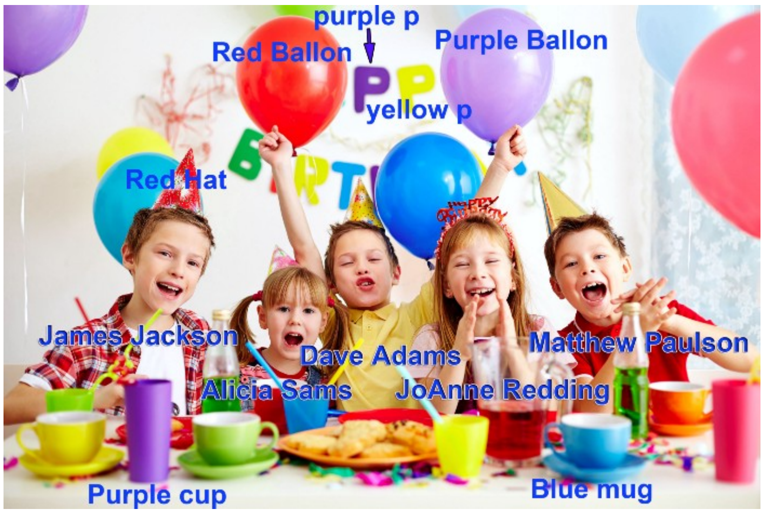


Figure 1: Object recognition of an image of a birthday party, from the website Medium [3].

To achieve objects and context recognition of an image, "**Bag Of Visual Words**" and **binary classifiers** have been used. A variable number of most common hashtags used with main hashtags are saved. For this report, there are 20 hashtags, ten main hashtags and ten selected after analyzing the usage of the main one.

For each hashtag, a binary classifier is used to determine if that particular tag is correlated to the image. There are two classifiers, one based on the **Logistic Regression** and the other one on the **K-Nearest Neighbors** technique.

1.1 Objectives

The objectives can be summarised in:

- Create a dataset of images acquired by hashtags.
- Create a list of most common hashtags, starting from the first hashtags.
- Train a binary classifier for each hashtag.
- Given an input image, return a list of hashtags executing the binary classifiers.

2 Images Mining

On the Internet, it is possible to find some images dataset. However, for this project has been decided to create its own images dataset from a Social Media, Instagram.

The first step is to choose the main hashtags, which define the category of the dataset's images.

For this report, the main hashtags are:

1. friends
2. food
3. animal
4. sea
5. sky
6. nature

7. street
8. people
9. car
10. building

2.1 Tools

Instagram has its API [4].
The basic API version, called **Instagram Basic Display API**, does not allow users to get hashtagged media. Therefore, it would have been necessary to use the advanced version, called **Instagram Graph API**, but it needs a business or creator account.

For this project, it has been sufficient to use a **scraper** that allows users to easily get images and lists of hashtags.

The scraper in question has been founded on GitHub, and it is created by the user "arc298". On the GitHub page of the project [5], it is present a guide on how to use the scraper. Some of the steps done are described below.

It is possible to install the program by execution on the CLI the following command:

```
$ pip install instagram-scraper
```

It is possible to specify media types to scrape with the option "**-t**", in this case, "image".
"– **media-metadata**" is used to saves the media metadata associated with the user's posts as a .json file.
"**-m**" to define the maximum number of items to scrape.
"– **tag**" to specify the hashtag for media.

For each tag, to effectuate the images mining, a similar command should be executed:

```
$ instagram-scraper -t image --media-metadata -m 1200 --tag sea
```

After the execution over the ten tags:

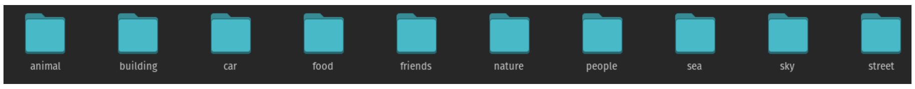


Figure 2: The directory structure of the dataset

2.2 Raw data

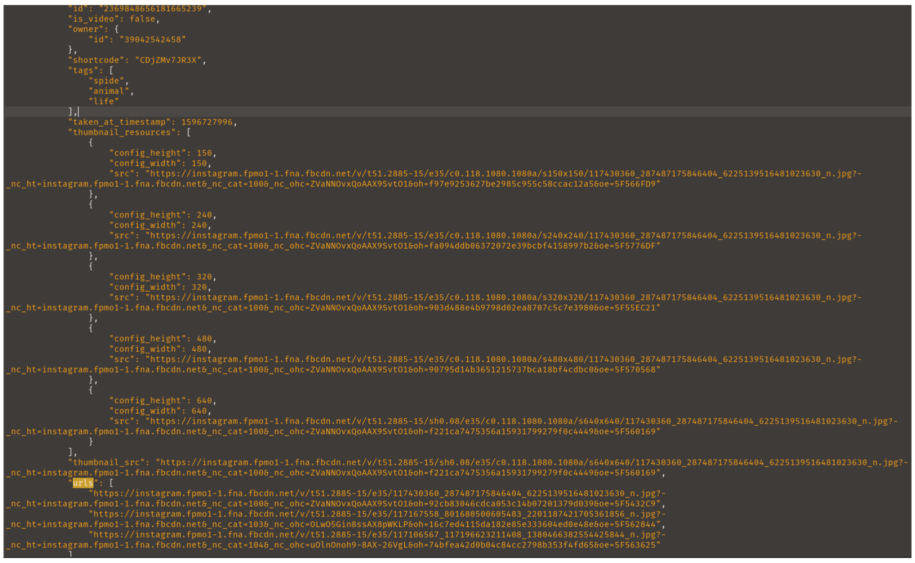
It is possible to notice that there are also .mp4 files, looking at the content of one folder closely, because the scraper downloads all the parts of a post. Besides, the JSON files contain much more information than it is needed(see Figure 3).

In the .JSON files, each picture is defined with the following attribute:

- comments_disabled
- dimensions
- display_url
- edge_media_preview_like
- edge_media_to_caption

- edge_media_to_comment
- id
- is_video
- owner
- shortcode
- tags
- taken_at_timestamp
- thumbnail_resources
- thumbnail_src
- urls

The only information needed is "tags" that contains the list of hashtags of the post and "urls" that contains the list of the links to the picture(s) of the post.



```
            "id": "2369848656181665239",
            "is_video": false,
            "owner": {
                "id": "39042542458"
            },
            "shortcode": "CDjZMv7JR3X",
            "tags": [
                "spide",
                "animal",
                "life"
            ],
            "taken_at_timestamp": 1596727996,
            "thumbnail_resources": [
                {
                    "config_height": 150,
                    "config_width": 150,
                    "src": "https://instagram.fpmo1-1.fna.fbcdn.net/v/t51.2885-15/e35/c0.118.1080.1080a/s150x150/117430360_287487175846404_6225139516481023630_n.jpg?-
_nc_ht=instagram.fpmo1-1.fna.fbcdn.net&_nc_cat=100&_nc_ohc=ZVaNNOvxQoAAX9SvtO1&oh=f97e9253627be2985c955c58ccac12a5&oe=5F566FD9"
                },
                {
                    "config_height": 240,
                    "config_width": 240,
                    "src": "https://instagram.fpmo1-1.fna.fbcdn.net/v/t51.2885-15/e35/c0.118.1080.1080a/s240x240/117430360_287487175846404_6225139516481023630_n.jpg?-
_nc_ht=instagram.fpmo1-1.fna.fbcdn.net&_nc_cat=100&_nc_ohc=ZVaNNOvxQoAAX9SvtO1&oh=fa094ddb06372072e39bcbf4158997b2&oe=5F5776DF"
                },
                {
                    "config_height": 320,
                    "config_width": 320,
                    "src": "https://instagram.fpmo1-1.fna.fbcdn.net/v/t51.2885-15/e35/c0.118.1080.1080a/s320x320/117430360_287487175846404_6225139516481023630_n.jpg?-
_nc_ht=instagram.fpmo1-1.fna.fbcdn.net&_nc_cat=100&_nc_ohc=ZVaNNOvxQoAAX9SvtO1&oh=903d488e4b9798d02ea8707c5c7e3980&oe=5F55EC21"
                },
                {
                    "config_height": 480,
                    "config_width": 480,
                    "src": "https://instagram.fpmo1-1.fna.fbcdn.net/v/t51.2885-15/e35/c0.118.1080.1080a/s480x480/117430360_287487175846404_6225139516481023630_n.jpg?-
_nc_ht=instagram.fpmo1-1.fna.fbcdn.net&_nc_cat=100&_nc_ohc=ZVaNNOvxQoAAX9SvtO1&oh=90795d14b3651215737bca18bf4cdbc0&oe=5F570568"
                },
                {
                    "config_height": 640,
                    "config_width": 640,
                    "src": "https://instagram.fpmo1-1.fna.fbcdn.net/v/t51.2885-15/sh0.08/e35/c0.118.1080.1080a/s640x640/117430360_287487175846404_6225139516481023630_n.jpg?-
_nc_ht=instagram.fpmo1-1.fna.fbcdn.net&_nc_cat=100&_nc_ohc=ZVaNNOvxQoAAX9SvtO1&oh=f221ca7475356a15931799279f0c4449&oe=5F560169"
                }
            ],
            "thumbnail_src": "https://instagram.fpmo1-1.fna.fbcdn.net/v/t51.2885-15/sh0.08/e35/c0.118.1080.1080a/s640x640/117430360_287487175846404_6225139516481023630_n.jpg?-
_nc_ht=instagram.fpmo1-1.fna.fbcdn.net&_nc_cat=100&_nc_ohc=ZVaNNOvxQoAAX9SvtO1&oh=f221ca7475356a15931799279f0c4449&oe=5F560169",
            "urls": [
                "https://instagram.fpmo1-1.fna.fbcdn.net/v/t51.2885-15/e35/117430360_287487175846404_6225139516481023630_n.jpg?-
_nc_ht=instagram.fpmo1-1.fna.fbcdn.net&_nc_cat=100&_nc_ohc=ZVaNNOvxQoAAX9SvtO1&oh=92cb83046cdca053c14b07201379d039&oe=5F5432C9",
                "https://instagram.fpmo1-1.fna.fbcdn.net/v/t51.2885-15/e35/117167558_801680500605483_2201187421705361856_n.jpg?-
_nc_ht=instagram.fpmo1-1.fna.fbcdn.net&_nc_cat=103&_nc_ohc=OLwO5Gin8ssAX8pWKLP&oh=16c7ed4115da182e85e333604ed0e48e&oe=5F562844",
                "https://instagram.fpmo1-1.fna.fbcdn.net/v/t51.2885-15/e35/117106567_117196623211408_1380466382554425844_n.jpg?-
_nc_ht=instagram.fpmo1-1.fna.fbcdn.net&_nc_cat=104&_nc_ohc=uOlnOnoh9-8AX-26VgL&oh=74bfea42d0b04c84cc2798b353f4fd65&oe=5F563625"
            ]
```

Figure 3: Some attributes in a JSON file

Notice that "urls" could have more than one value; this happens when a post contains more pictures. "urls" values include the name under which the images are saved in the folder.

The "tags" list could include some hashtags that have been decided to be filtered.

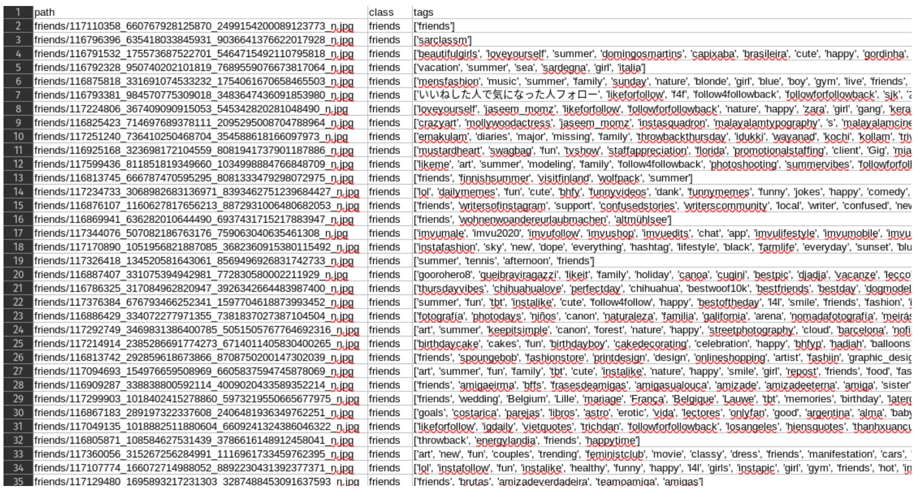
The observations above suggest that data in the current step is in the state of "**raw data**," or rather, data that needs to be processed.

3 Dataset creation

Instead of having many .json files, it is convenient to have a single file, representing the project's database.

This file must include the necessary information to read the images, know the main tag that led to obtaining the corresponding images, and the list of each image's tags.

The file extension chosen is .csv, in order to make easier the reading process of the dataset using the pandas library [6].



1	path	class	tags
2	friends/117110358_660767928125870_2499154200089123773_n.jpg	friends	['friends']
3	friends/116796396_635418033845931_9036641376622017928_n.jpg	friends	['sarclassm']
4	friends/116791532_175573687522701_5464715492110795818_n.jpg	friends	['beautifulgirls', 'loveyourself', 'summer', 'domingosmartins', 'capixaba', 'brasileira', 'cute', 'happy', 'gordinha',
5	friends/116792328_950740202101819_7689559076673817064_n.jpg	friends	['vacation', 'summer', 'sea', 'sardegna', 'girl', 'italia']
6	friends/116875818_331691074533232_1754061670658465503_n.jpg	friends	['mensfashion', 'music', 'summer', 'family', 'sunday', 'nature', 'blonde', 'girl', 'blue', 'boy', 'gym', 'live', 'friends',
7	friends/116793381_984570775309018_3483647436091853980_n.jpg	friends	['いいねした人で気になった人フォロー', 'likeforfollow', 'f4f', 'follow4followback', 'followforfollowback', 'sjk', '
8	friends/117224806_367409090915053_545342820281048490_n.jpg	friends	['loveyourself', 'jaseem_momz', 'likeforfollow', 'followforfollowback', 'nature', 'happy', 'zara', 'girl', 'gang', 'kera
9	friends/116825423_714697689378111_2095295008704788964_n.jpg	friends	['crazyart', 'mollywoodactress', 'jaseem_momz', 'instasquadron', 'malayalamtypography', 's', 'malayalamcine
10	friends/117251240_736410250468704_354588618166097973_n.jpg	friends	['ernakulam', 'diaries', 'major', 'missing', 'family', 'throwbackthursday', 'idukki', 'wayanad', 'kochi', 'kollam', 'tri
11	friends/116925168_323698172104559_8081941737901187886_n.jpg	friends	['mustardheart', 'swagbag', 'fun', 'tvshow', 'staffappreciation', 'florida', 'promotionalstaffing', 'client', 'Gig', 'mia
12	friends/117599436_811851819349660_1034998884766848709_n.jpg	friends	['likeme', 'art', 'summer', 'modeling', 'family', 'follow4followback', 'photoshooting', 'summervibes', 'followforfoll
13	friends/116813745_666787470595295_8081333479298072975_n.jpg	friends	['friends', 'finnishsummer', 'visitfinland', 'wolfpack', 'summer']
14	friends/117234733_3068982683136971_8393462751239684427_n.jpg	friends	['lol', 'dailymemes', 'fun', 'cute', 'bhfy', 'funnyvideos', 'dank', 'funnymemes', 'funny', 'jokes', 'happy', 'comedy',
15	friends/116876107_1160627817656213_8872931006480682053_n.jpg	friends	['friends', 'writersofinstagram', 'support', 'confusedstories', 'writerscommunity', 'local', 'writer', 'confused', 'new
16	friends/116869941_636282010644490_6937431715217883947_n.jpg	friends	['friends', 'wohnenwoandereurlaubmachen', 'altmühlsee']
17	friends/117344076_507082186763176_759063040635461308_n.jpg	friends	['imvumale', 'imvu2020', 'imvufollow', 'imvushop', 'imvuedits', 'chat', 'app', 'imvulifestyle', 'imvumobile', 'imvu
18	friends/117170890_1051956821887085_3682360915380115492_n.jpg	friends	['instafashion', 'sky', 'new', 'dope', 'everything', 'hashtag', 'lifestyle', 'black', 'farmlife', 'everyday', 'sunset', 'blu
19	friends/117326418_134520581643061_8569496926831742733_n.jpg	friends	['summer', 'tennis', 'afternoon', 'friends']
20	friends/116887407_331075394942981_772830580002211929_n.jpg	friends	['goorohero8', 'queibraviragazzi', 'likeit', 'family', 'holiday', 'canoa', 'cugini', 'bestpic', 'djadja', 'vacanze', 'lecco'
21	friends/116786325_317084962820947_3926342664483987400_n.jpg	friends	['thursdayvibes', 'chihuahualove', 'perfectday', 'chihuahua', 'bestwoof10k', 'bestfriends', 'bestday', 'dogmodel
22	friends/117376384_676793466252341_1597704618873993452_n.jpg	friends	['summer', 'fun', 'tbt', 'instalike', 'cute', 'follow4follow', 'happy', 'bestoftheday', 'l4l', 'smile', 'friends', 'fashion', 'i
23	friends/116886429_334072277971355_7381837027387104504_n.jpg	friends	['fotografía', 'photodays', 'niños', 'canon', 'naturaleza', 'familia', 'galifornia', 'arena', 'nomadafotografía', 'meirás
24	friends/117292749_3469831386400785_5051505767764692316_n.jpg	friends	['art', 'summer', 'keepitsimple', 'canon', 'forest', 'nature', 'happy', 'streetphotography', 'cloud', 'barcelona', 'nofi
25	friends/117214914_2385286691774273_6714011405830400265_n.jpg	friends	['birthdaycake', 'cakes', 'fun', 'birthdayboy', 'cakedecorating', 'celebration', 'happy', 'bhfyp', 'hadiah', 'balloons'
26	friends/116813742_292859618673866_8708750200147302039_n.jpg	friends	['friends', 'spoungebob', 'fashionstore', 'printdesign', 'design', 'onlineshopping', 'artist', 'fashiin', 'graphic_desig
27	friends/117094693_154976659508969_6605837594745878069_n.jpg	friends	['art', 'summer', 'fun', 'family', 'tbt', 'cute', 'instalike', 'nature', 'happy', 'smile', 'girl', 'repost', 'friends', 'food', 'fas
28	friends/116909287_338838800592114_4009020433589352214_n.jpg	friends	['friends', 'amigaeirma', 'bffs', 'frasesdeamigas', 'amigasualouca', 'amizade', 'amizadeeterna', 'amiga', 'sister'
29	friends/117299903_1018402415278860_5973219550665677975_n.jpg	friends	['friends', 'wedding', 'Belgium', 'Lille', 'mariage', 'França', 'Belgique', 'Lauwe', 'tbt', 'memories', 'birthday', 'later
30	friends/116867183_289197322337608_2406481936349762251_n.jpg	friends	['goals', 'costarica', 'parejas', 'libros', 'astro', 'erotic', 'vida', 'lectores', 'onlyfan', 'good', 'argentina', 'alma', 'baby
31	friends/117049135_1018882511880604_6609241324386046322_n.jpg	friends	['likeforfollow', 'igdaily', 'vietquotes', 'trichdan', 'followforfollowback', 'losangeles', 'hiensquotes', 'thanhxuancu
32	friends/116805871_108584627531439_3786616148912458041_n.jpg	friends	['throwback', 'energylandia', 'friends', 'happytime']
33	friends/117360056_315267256284991_1116961733459762395_n.jpg	friends	['art', 'new', 'fun', 'couples', 'trending', 'feministclub', 'movie', 'classy', 'dress', 'friends', 'manifestation', 'cars', '
34	friends/117107774_166072714988052_8892230431392377371_n.jpg	friends	['lol', 'instafollow', 'fun', 'instalike', 'healthy', 'funny', 'happy', 'l4l', 'girls', 'instapic', 'girl', 'gym', 'friends', 'hot', 'in
35	friends/117129480_1695893217231303_3287488453091637593_n.jpg	friends	['friends', 'brutas', 'amizadeverdadeira', 'teamoamiga', 'amigas']

Figure 4: .csv file structure

3.1 Data transformation

As explained in the section "Raw Data", some information is useless for the models' training process. It is convenient to define a transformation function to manipulate data, following the line of reasoning discussed before.

The data transformation has permitted to save memory, deleting unnecessary files, and more comfortable to manage, considering that the .csv file has only three columns instead of all JSON files' attributes. JSON files have been read using the json library [7].

```python
# starting from a path to the data_set folder that contains some folders,
# create a CSV with 3 columns(the path of the image, class and relative list of tags)
def csv_creation(data_set_path, main_tags_path):
    df = pd.DataFrame(columns=['path', 'class', 'tags'])
    target_flags = open(main_tags_path).read().splitlines()

    for tag in target_flags:
        path = data_set_path + os.path.sep + tag + os.path.sep + tag + '.json'
        count = 0

        with open(path) as json_file:
            data = json.load(json_file)
            for p in data['GraphImages']:
                if count >= 1000:
                    break

                if len(p['urls']) == 0:
                    continue

                url = p['urls'][0]
                result = re.search('/[0-9](.*)\.jpg', url)
                if result is None:
                    continue

                url_final = tag + result.group(0)

                try:
                    tags = p['tags']
                except KeyError:
                    # this tuple doesn't contain any 'tags' attribute
                    continue
```

```
32
33                  if len(tags) == 0:
34                      tags = [tag]
35
36                  tags = clean_tags(tags)
37
38                  new_row = {'path': url_final, 'class': tag, 'tags': tags}
39                  df = df.append(new_row, ignore_index=True)
40                  count += 1
41
42          print('\'' + tag + '\'' + " tag completed")
43
44      # index=False because I don't need to save a column with indexes
45      df.to_csv(DATASET_NAME_CSV, index=False)
```

Notice line 14 of the code above, for this project, has been decided to save only 1000 images for each tag: 10k images in total.

In line 36, the tags are filtered by calling the function "clean_tags", removing some of the most used hashtags on Instagram [1].

```
def clean_tags(tags):
    stop_words = ['love', 'photography', 'instagood', 'travel', 'beautiful', 'style',
         'follow', 'photooftheday',
                  'picoftheday', 'instagram', 'photo', 'naturephotography', 'life',
                  'instadaily', 'travelphotography']
    tags_cleaned = list(filter(lambda tag: tag not in stop_words, tags))
    return tags_cleaned
```

After the .csv file has been saved, it is possible to read it in a coincide way, using the pandas library.

```
# return a DataFrame filled by a CSV, located in the path
def get_data_csv(path):
    data = pd.read_csv(path)
    return data
```

4 Training and Testing sets

Before the training of the models, it is necessary to create a training set for each tag. After the training process, to evaluate the performance of the machine learning algorithm, a testing set is also necessary. These sets are necessary to train and evaluate binary classifiers; they necessitate binary labels(e.g., 'tag' and 'not tag'), see Figure 8.

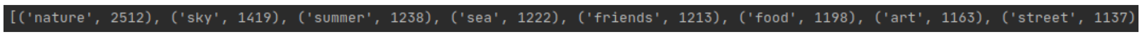
The tags for which the training and testing sets are necessary, need to be found. Therefore, before creating the sets, all the tags need to be processed to find the most commons.

Some functions have been useful for a concise and functioning code, e.g., **literal_eval** from the ast module [8] and the class **Counter** from the collections module [9].

```
# return a list of tags as a Series object
def get_tags_from_csv(path_csv):
    data = get_data_csv(path_csv)
    return data['tags']


```

```
 7 # return a dictionary with the frequency of each tag in the tags Series. Each element of tags is a string of tags.
 8 # Using literal_eval it is possible to convert a string into a list obtaining a list of tags
 9 def tags_frequency(tags, num_tags):
10     # Creating a list of tags using the list comprehension
11     list_tags = [tag for index in range(0, len(tags)) for tag in literal_eval(tags[index])]
12     counter = Counter(list_tags)
13     most_common_counter = counter.most_common(num_tags)
14     most_common_tags = [tag for tag, value in most_common_counter]
15     return most_common_tags
16 
17 
18 # It creates a file containing each tag in tags, one per line
19 def create_file_most_common_tags(path, tags):
20     with open(path, "w") as output:
21         for tag in tags:
22             output.write(str(tag) + '\n')
```

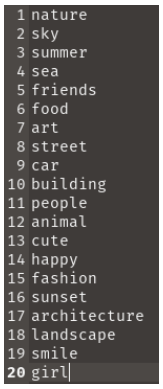
In line 13, *most_ common* function returns a list of the most common elements and their counts from the most common to the least(see Figure 5).



```
[('nature', 2512), ('sky', 1419), ('summer', 1238), ('sea', 1222), ('friends', 1213), ('food', 1198), ('art', 1163), ('street', 1137)
```

Figure 5: Example of the content of *most_ common_ counter*.

Starting from the main hashtags of section 2, it led to other hashtags (see Figure 6).



```
 1 nature
 2 sky
 3 summer
 4 sea
 5 friends
 6 food
 7 art
 8 street
 9 car
10 building
11 people
12 animal
13 cute
14 happy
15 fashion
16 sunset
17 architecture
18 landscape
19 smile
20 girl
```

Figure 6: The 20 most common hashtags, obtained starting from the main hashtags

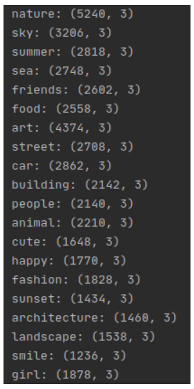
4.1 Dimensions and split criteria

The training and testing set dimension has been set up depending on the number of images with a specific tag. Hence, most frequent hashtags have a broader training and testing set (see Figure 7). Precisely, each training set contains all the pictures with that tag as "positive" and the same number of examples without that tag as "negative." Instead, each testing set contains all the examples with that tag as "positive" and three times more examples without that tag as "negative."

```
# It creates a set for a tag in the file path_tags. The set's balance depends on the balance factor
def set_creation(data, tag, balance=1):
    data_set = pd.DataFrame()

    # it shuffles the dataframe
    temp_data = data.copy()
    data_shuffle = temp_data.sample(frac=1)
```

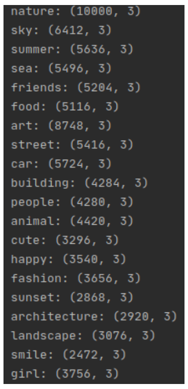
```
 8
 9      for i in range(0, len(temp_data)):
10          if tag in temp_data.iloc[i]['tags']:
11              new_row = temp_data.iloc[i].copy()
12              new_row['class'] = tag
13              data_set = data_set.append(new_row)
14
15      num_rows = len(data_set)
16      counter = 0
17
18      for i in range(0, len(data_shuffle)):
19          if counter >= num_rows * balance:
20              break
21          if tag not in data_shuffle.iloc[i]['tags']:
22              new_row = data_shuffle.iloc[i].copy()
23              new_row['class'] = 'not ' + tag
24              data_set = data_set.append(new_row)
25              counter += 1
26
27      return data_set
28
29
30  # creating a training and testing sets for each tag in the file path_tags
31  def training_testing_set_creation(path_tags):
32      data = get_data_csv(DATASET_NAME_CSV)
33      target_flags = open(path_tags).read().splitlines()
34
35      for tag in target_flags:
36          training_set = set_creation(data, tag, balance=1)
37          save_object(TRAINING_SET_FOLDER + os.path.sep + tag, training_set)
38
39          testing_set = set_creation(data, tag, balance=3)
40          save_object(TESTING_SET_FOLDER + os.path.sep + tag, testing_set)
41          print(tag + ' completed')
```

Notice the line 7 of the code above that **shuffles** the dataset. This step is particularly crucial because it leads to preventing the presence of any bias in the creation of the sets. Indeed, being the dataset(the .csv file) immutable, without shuffling it, most of the examples of the sets could be the same, not simulating a "real world" execution.



```
nature: (5240, 3)
sky: (3206, 3)
summer: (2818, 3)
sea: (2748, 3)
friends: (2602, 3)
food: (2558, 3)
art: (4374, 3)
street: (2708, 3)
car: (2862, 3)
building: (2142, 3)
people: (2140, 3)
animal: (2210, 3)
cute: (1648, 3)
happy: (1770, 3)
fashion: (1828, 3)
sunset: (1434, 3)
architecture: (1460, 3)
landscape: (1538, 3)
smile: (1236, 3)
girl: (1878, 3)
```

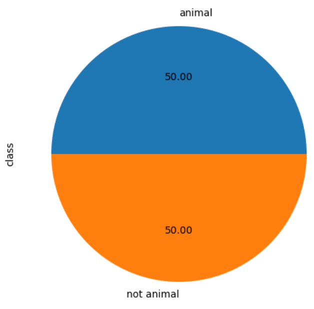
(a) Training sets



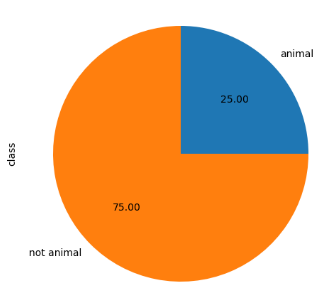
```
nature: (10000, 3)
sky: (6412, 3)
summer: (5636, 3)
sea: (5496, 3)
friends: (5204, 3)
food: (5116, 3)
art: (8748, 3)
street: (5416, 3)
car: (5724, 3)
building: (4284, 3)
people: (4280, 3)
animal: (4420, 3)
cute: (3296, 3)
happy: (3540, 3)
fashion: (3656, 3)
sunset: (2868, 3)
architecture: (2920, 3)
landscape: (3076, 3)
smile: (2472, 3)
girl: (3756, 3)
```

(b) Testing sets

Figure 7: The shape of the sets



(a) Training set



(b) Testing set

Figure 8: The balance of the sets for the tag "animal"

5 Bag of Visual Words model

The Bag of Visual Words (BoVW) model can be used to implement the image classification, namely, the scope of this project, by treating image features as words. Besides, using the BoVW model, it's possible to work with images of different resolution, due to the fixed-length representation, without having to make the dataset uniform.

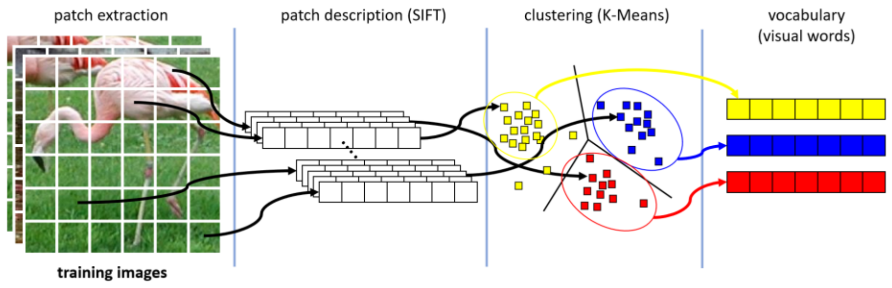


Figure 9: Bag Of Visual Words model creation process

5.1 Patch extraction and description

To extract and describe patches has been used cyvlfeat, a Python wrapper to the VLFeat library. It is possible to define two parameters for the function dsift: *step* and *size*.
The sampling step is defined by the *step* value and the bin size by the *size* value (see Figure 10). These two values are used to increasing or decreasing the number of patches extracted.

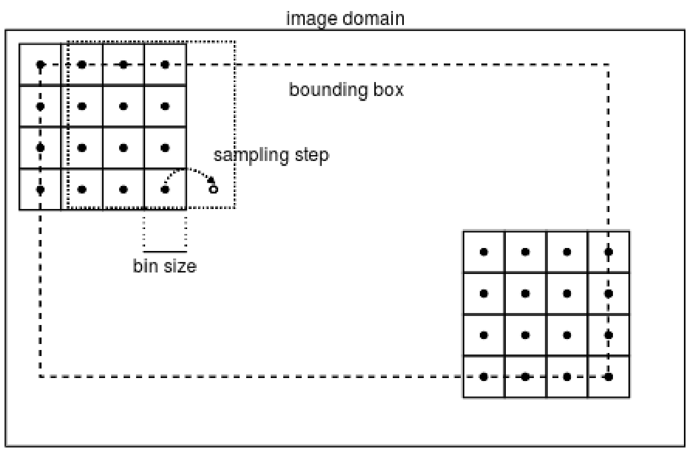


Figure 10: Dense SIFT descriptor geometry, from the VLFeat documentation [10]

Considering the number and the dimension of the sets, for the report, the *step* parameter has been set to 60, in order to decrease the number of patches.

It is possible to set another parameter of the function dsift, that is *fast*. The parameter *fast* can assume two values, True or False(default).
Setting *fast* to True allows increasing the feature transform **speed**, resulting from 30 to 70 times faster than SIFT [11]. In order to the benefit of the lowest error percentage and the best speedup, the *size* has been set to 5 (see Figure 11).

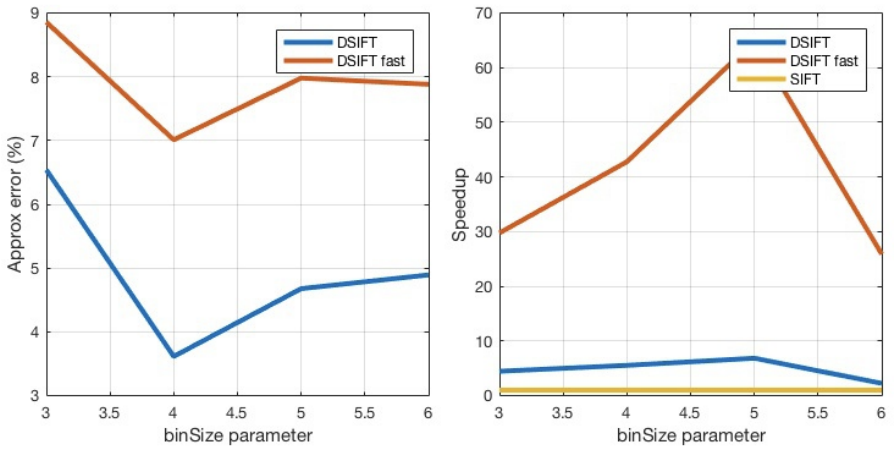


Figure 11: Accuracy and speedup comparison between DSIFT and DSIFT fast, from the VLFeat documentation [11]

```
1  # it extracts and describes all the patches of the images in data using the dsift function
2  def extract_and_describe(data, size=5, step=STEP):
3      descriptors = []
4      for i, row in tqdm(data.iterrows(), "Extracting/Describing Patches", total=len(data)):
5          path = DATA_SET_FOLDER + os.path.sep + row['path']
6          im = io.imread(path, as_gray=True)
7  
8          _, description = dsift(im, size=size, step=step, fast=True)
9          descriptors.append(description)
10 
11     return np.vstack(descriptors)
```

Line 11 is used to concatenate the features of two or more images; thus, it is possible to extract and describe all the images of a set at one time.

5.2 Visual Vocabulary

After having extracted and described the patches of the training set, it is time for the "visual words vocabulary" created by clustering the features.
The training descriptors can be clustered using the MiniBatchKMeans from scikit- learn library [12], which is an implementation of the K-Means algorithm optimized to operate over vast sets of data.

The vocabulary size has been set to 500 words, which has been shown to be an appropriate value [13].
The k_means is fitted using the training set description and then used to associate a Bag of Visual Words to a new picture using the method predict.

```
# it describes an image and then return the bag of visual words
def load_and_describe(filename, size=5, step=STEP):
    im = io.imread(filename, as_gray=True)
    _, descriptors = dsift(im, size=size, step=step, fast=True)
    tokens = k_means.predict(descriptors)
    return tokens
```

6 Classification

Before the classification process, a vectorization of the BoVW and eventually a normalization, are necessary.

As a pre-processing step, a weighting scheme has been used based on term-frequency inverse document-frequency (**TF-IDF**) to scale up the rare visual words while damping the effect of the frequent visual words.

Vectorization and the normalization have been applied using *TfidfVectorizer* from the sklearn library [14].

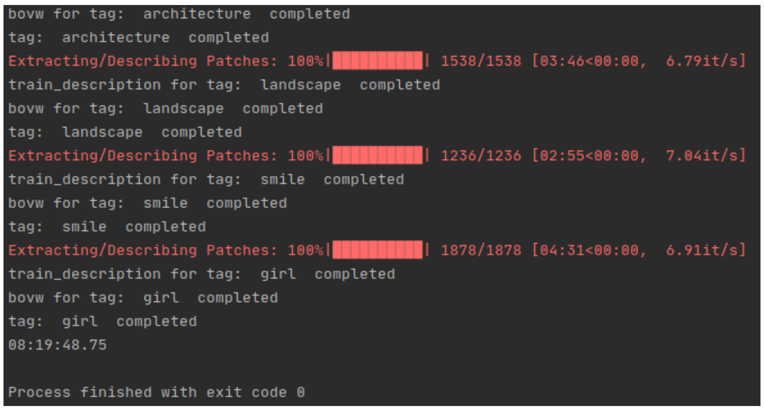
The *TfidfVectorizer* should transform the training and testing set as a document-term matrix. This is possible using the method transform but only after learning the vocabulary from the training set.
Considering that the training set should be used to fit, and then it should be transformed, it is reasonable to use the method *fit_transform*, which does both things in one command.

```
1  # it creates and saves the bag of visual words for train and test sets applying tf-idf
       normalization
2  def bovw_normalized_creation(train_descriptions, tag, train_set, test_set):
3      # k_means needs to be fitted before executing load_and_describe function
4      k_means.fit(train_descriptions)
5      save_object(K_MEANS_FOLDER + os.path.sep + tag + '_kmeans', k_means)
6
7      tfidf = TfidfVectorizer(tokenizer=load_and_describe,
           vocabulary=range(NUM_VOCABULARY), use_idf=True)
8
9      x_train = tfidf.fit_transform(DATA_SET_FOLDER + os.path.sep + train_set['path'])
10     save_object(BOVW_FOLDER + os.path.sep + tag + '_train', x_train)
11
12     x_test = tfidf.transform(DATA_SET_FOLDER + os.path.sep + test_set['path'])
13     save_object(BOVW_FOLDER + os.path.sep + tag + '_test', x_test)
14
15     save_object(TFIDF_FOLDER + os.path.sep + tag, tfidf)
```

The *TfidfVectorizer*, in line 7, has been set up with the same vocabulary size of the K-Mean. Passing the function *load_and_describe* as tokenizer leads the *TfidfVectorizer* to load all the images, extract, and describe patches.
It is possible to define the *norm parameter*, which has as a default value the *l2* normalization. It has not been specified because the *l2* normalization has been preferred over the *l1* one for the lower computational load required [15].

```
def classifiers_creation(path_tags):
    target_flags = open(path_tags).read().splitlines()

    for tag in target_flags:
        train_set = load_object(TRAINING_SET_FOLDER + os.path.sep + tag)
        test_set = load_object(TESTING_SET_FOLDER + os.path.sep + tag)
        y_train = train_set['class']
        y_test = test_set['class']

        train_descriptions = extract_and_describe(train_set)
        save_object(TRAINING_DESCRIPTORS + os.path.sep + tag, train_descriptions)

        print("train_description for tag: ", tag, " completed")
        bovw_normalized_creation(train_descriptions, tag, train_set, test_set)
        print("bovw for tag: ", tag, " completed")
        x_train = load_object(BOVW_FOLDER + os.path.sep + tag + '_train')
        x_test = load_object(BOVW_FOLDER + os.path.sep + tag + '_test')

        lr = logistic_regression_fitting(x_train, y_train)
        save_object(LR_FOLDER + os.path.sep + tag, lr)

        # creating the validation set
        x_train, x_valid, y_train, y_valid = train_test_split(x_test, y_test,
            test_size=0.30)

        best_k = hyperparameter_optimization_knn(x_valid, y_valid)
        knn = nearest_neighbor_fitting(x_train, y_train, best_k)
        save_object(KNN_FOLDER + os.path.sep + tag, knn)
```

The above segment of code may take a long time. In the case of this report setting, with an Intel i5-8400 CPU, almost 8 hours and a half (see Figure 12).



```
bovw for tag:  architecture  completed
tag:  architecture  completed
Extracting/Describing Patches: 100%|██████████| 1538/1538 [03:46<00:00,  6.79it/s]
train_description for tag:  landscape  completed
bovw for tag:  landscape  completed
tag:  landscape  completed
Extracting/Describing Patches: 100%|██████████| 1236/1236 [02:55<00:00,  7.04it/s]
train_description for tag:  smile  completed
bovw for tag:  smile  completed
tag:  smile  completed
Extracting/Describing Patches: 100%|██████████| 1878/1878 [04:31<00:00,  6.91it/s]
train_description for tag:  girl  completed
bovw for tag:  girl  completed
tag:  girl  completed
08:19:48.75

Process finished with exit code 0
```

Figure 12: Execution time needed to fit and classify all the datasets for each tag

6.1 Logistic Regression

One of the techniques used for the classification is the **Logistic Regression**, which can be implemented in python importing the *LogisticRegression* class from the sklearn library [16].

```
# fitting process for Logistic Regression
def logistic_regression_fitting(x_train, y_train):
    lr = LogisticRegression(multi_class='ovr', solver='sag')
    lr.fit(x_train, y_train)
    return lr
```

The *multi_ class* parameter has been set to *ovr*, in order to fit a binary problem for each label. Among the various solvers, *sag* has been chosen because it is recommended for large datasets.

6.2 K-Nearest Neighbors

Besides the Logistic Regression, also the **K-Nearest Neighbors** technique has been used. The K-Nearest Neighbors can be used in python by importing the *KNeighborsClassifier* class from the sklearn library [17].

```
# fitting process for Nearest Neighbor
def nearest_neighbor_fitting(x_train, y_train, k=1):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train, y_train)
    return knn
```

One of the parameters that can be set is *n_ neighbors*, which is the number of neighbors to use for the classification process.
n_ neighbors is a **hyperparameter** that should be decided before fitting the training set.

6.2.1 Hyperparameter choice

One way to set the hyperparameter is trying to optimize the performance measure on a validation set, trying some *n_ neighbors* values.
The validation set is a third set, obtained from the training set (see lines 23-26 of the code in section 6).
For this report, the *n_ neighbors* values tested are from 1 to 9.

```
# it calculates the best hyperparameter for K-nearest neighbors classifier
def hyperparameter_optimization_knn(x_valid, y_valid):
    best_k = 1
    best_score = 0
    for k in range(1, 10):
        knn = nearest_neighbor_fitting(x_valid, y_valid, k)
        y_valid_pred = knn.predict(x_valid)
        score = f1_score(y_valid, y_valid_pred, average=None).mean()
        if score > best_score:
            best_score = score
            best_k = k

    return best_k
```

7 Performances

Before calculating the performance of the classifiers, it is necessary to choose a measure. For this report, considering that the test set is not balanced (see Figure 9),a measure that works appropriately with an unbalanced set is the *f1 score*. The *f1 score* can be calculated using the function *f1_ score* from the sklearn library [18].

7.1 Performance evaluation

It is possible to notice that the **K-Nearest Neighbors** have better performances, obtaining better *f1 scores* for every tag compared to the Logistic Regression (see Figure 13). However, the K-Nearest Neighbors is overall slower.

The Logistic Regression has seemed to be also sensible to the compliance of the dataset. Indeed, the tags such as girl, summer, fashion, and art, which usually are misused (e.g., it is difficult to define a fashion and what is not and the tag girl is used even by men to make posts more visible).

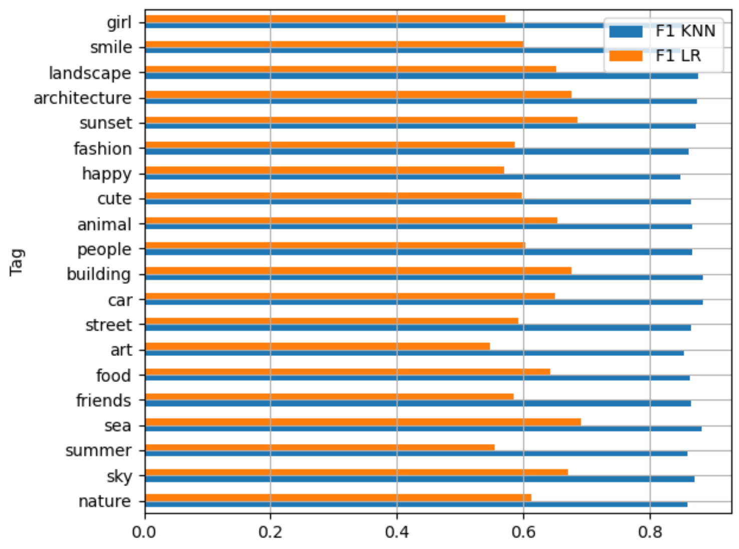
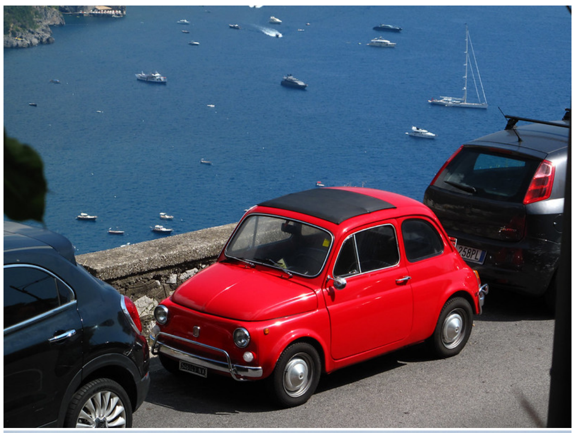


Figure 13: Plot with the performance measures of each classifier

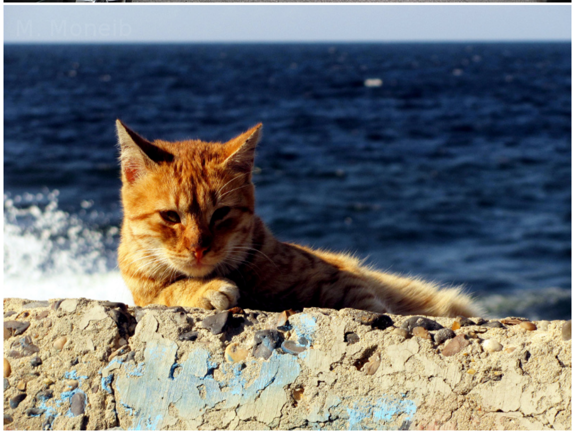
8 Conclusion

As a conclusion, some examples will show which hashtags are suggested by the algorithm on a few images.

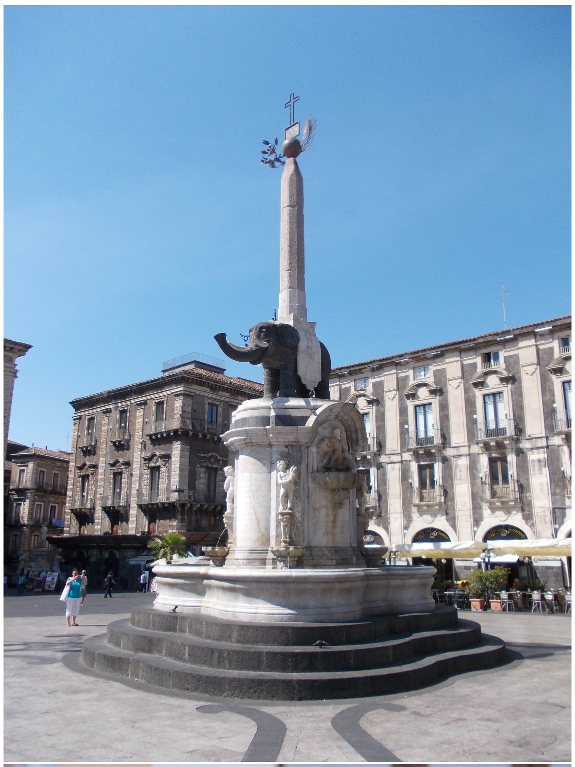
8.1 Examples



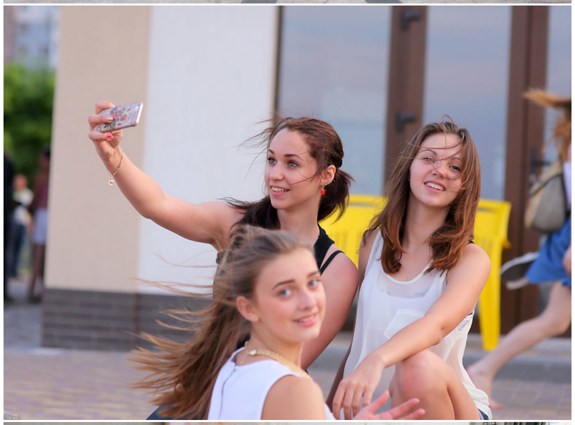
- lr: nature, sky, sea, car, landscape
- knn: nature, building, girl



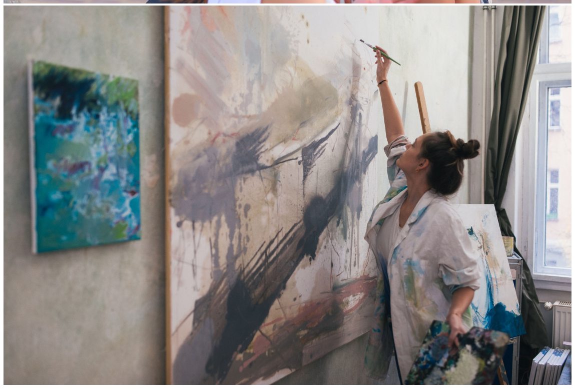
- lr: nature, summer, sea, car, animal
- knn: sea, street, animal, cute, happy, fashion, landscape



- lr: sky, sea, art, car, fashion, smile, girl
- knn: sea, art, building, fashion, architecture



- lr: nature, summer, friends, food, art, building, people, happy, fashion, architecture, girl
- knn: sky, summer, food, car, cute, architecture



- lr: sky, sea, friends, art, street, car, cute, fashion, smile, girl
- knn: summer, friends, building, cute, fashion, sunset, smile

References

[1] Instagram report 2020. https://mention.com/en/reports/instagram/hashtags/#5.

[2] J. Clement. Global social networks ranked by number of users 2020. https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/.

[3] Image recognition is not enough. https://medium.com/emergent-future/image-recognition-is-not-enough-293cd7d58004.

[4] Instagram API. https://developers.facebook.com/docs/instagram-api?locale=en_US.

[5] arc298, instagram-scraper. https://github.com/arc298/instagram-scraper.

[6] pandas - python data analysis library. https://pandas.pydata.org/.

[7] Json encoder and decoder. https://docs.python.org/3/library/json.html.

[8] ast — abstract syntax trees. https://docs.python.org/3/library/ast.html.

[9] collections — high-performance container datatypes. https://docs.python.org/2/library/collections.html.

[10] Dense scale invariant feature transform (dsift). https://www.vlfeat.org/api/dsift.html.

[11] Dense sift as a faster sift. https://www.vlfeat.org/overview/dsift.html.

[12] scikit-learn: machine learning in python. https://scikit-learn.org/.

[13] K. S. Sujatha, G. Karthiga, and Budai Shanmukha Vivek Vinod. Evaluation of bag of visual words for category level object recognition. pages 104–110, 2012.

[14] scikit-learn: Tfidfvectorizer. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html.

[15] L1 norms versus l2 norms. https://www.kaggle.com/residentmario/l1-norms-versus-l2-norms.

[16] scikit-learn: Logisticregression. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

[17] scikit-learn: Kneighborsclassifier. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html.

[18] scikit-learn: f1_score. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html.