گزارش پروژه درس پردازش زبان طبیعی

تهیه کنندگان:

شقایق مبشر – غزل زمانی نژاد – ملیکا نوبختیان

فهرست مطالب

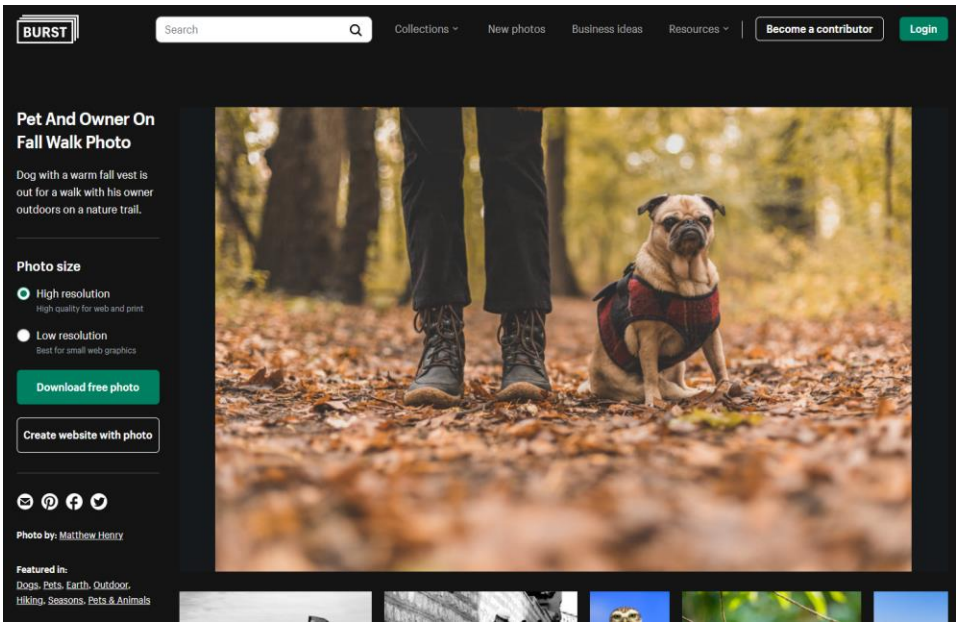
مقدمه

این پروژه تولید زیرنویس برای یک تصویر است. این کار در دو دسته‌بندی بینایی ماشین و پردازش زبان طبیعی قرار می‌گیرد. در این پروژه طی فازهای گوناگون سعی بر آن شد که ابتدا دیتاست لازم برای این تسک جمع‌آوری و سپس یک مدل آموزش دهیم.

۱- داده‌ها

برای جمع‌آوری تصاویر از روش web crawling استفاده شد. پس از بررسی میان سایت‌های معتبر از جمله instagram، pinterest، Wikipedia و غیره به سایت BURST رسیدیم. در این سایت تصاویر در دسته‌بندی‌های گوناگون همراه با یک title و کپشن به زبان انگلیسی وجود دارد.

به دلیل فیلتر بودن سایت، باز کردن صفحات آن و دانلود تصاویر بسیار طول می‌کشید. به همین دلیل از محیط google colab برای اجرای کد درون فایل NLPDatasetGenerator.ipynb و ساخت دیتاست استفاده کردیم. فایل با فرمت py. از این کد نیز وجود دارد اما پیشنهاد میکنیم از نوت بوک در محیط کولب استفاده کنید.



شکل ۱ – یک نمونه تصویر به همراه عنوان و کپشن در سایت BURST

۱-۱- فرمت داده‌های خام دیتاست

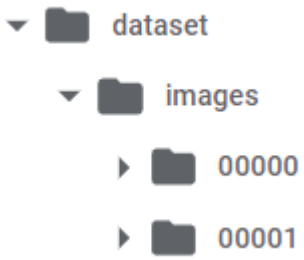
در مسیر دلخواهی که درون فایل نوت بوک مشخص می‌شود، یک فولدر raw ساخته می‌شود که در آن یک فایل dataset.json قرار دارد. این فایل شامل لیستی از دیکشنری‌هایی با کلیدهای id (از ۰ تا ۱۲۶۰۱)، title (عنوان صفحه ای از سایت burst که این داده قرار دارد)، image_url (لینک به تصویر داده در سایت burst) و caption (کپشن تصویر) است. به طور مثال از این صفحه اطلاعات زیر در قالب یک دیکشنری استخراج و در لیست ذخیره می‌شود:

```
{
        "id": 1,
        "title": "Cave Of Wonder And Lights Photo",
        "image_url": "https://burst.shopifycdn.com/photos/cave-of-wonder-and-
lights.jpg",
```

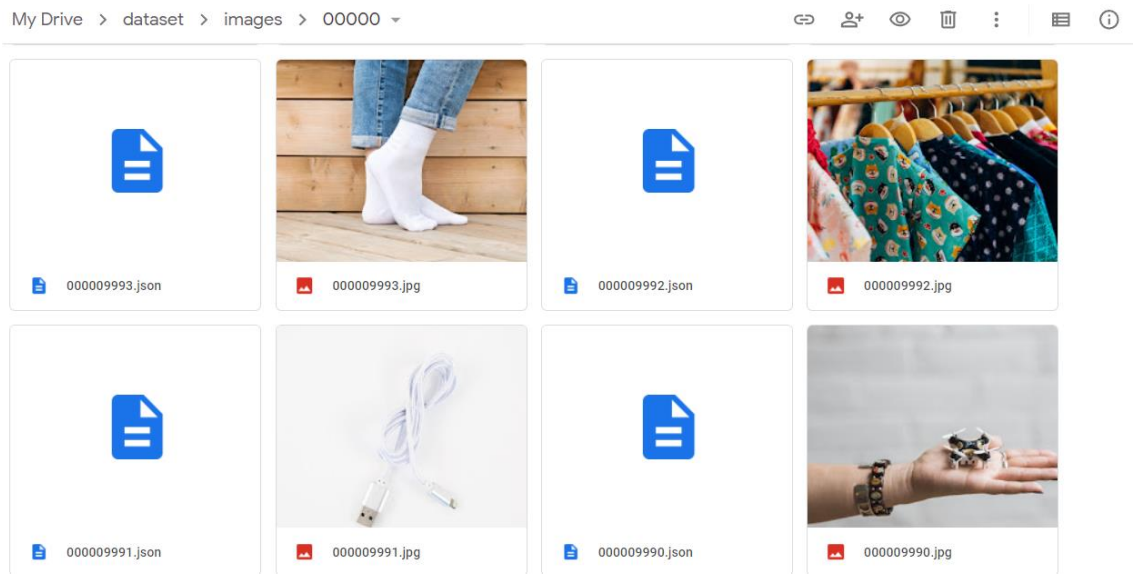
```
        "caption": "A man stands on a rock pillar in the middle of an underground
cave, a large ray of light casts down on him from an opening in the earth above."
    }
```

تعداد کل داده ها 12,602 می‌باشد.

همچنین در حالت پیش فرض با اجرای کد یک فولدر images در raw ساخته میشود که حاوی تصاویر دیتاست می‌باشد. به دلیل استفاده از کتابخانه img2dataset فرمت ذخیره تصاویر به این صورت است:



این کتابخانه هر ۱۰۰۰۰ تصویری که دانلود می‌کند را در یک فولدر با شماره گذاری از ۰ قرار می‌دهد. به این ترتیب تصاویر ۰ تا ۹۹۹۹ در فولدر 00000 و تصاویر ۱۰۰۰۰ تا ۱۲۶۰۱ در فولدر 00001 هستند. همراه با این دو فولدر دو فایل parquet و json هم برای نگهداری وضعیت دانلود موازی تصاویر و کنترل آن می‌سازد. هر کدام از تصاویر یک فایل json هم دارند که اطلاعات دانلود آن تصویر و جزئیاتش را داراست.



شکل ۲- نمونه ای از تصاویر ذخیره شده

همچنین هنگام جمع آوری دیتاست یک فایل unsuccessful_page_loads.txt نیز در فولدر raw ساخته می‌شود که url صفحاتی که نتوانسته باز کند را در هر خط دارد.

۲-۱- شرح کد

در ادامه کد درون فایل نوت بوک توضیح داده خواهد شد. فایل پایتون نیز همین توابع را دارد، تنها تفاوت در چند cell ابتداییست که برای نصب پکیج‌ها و قرار دادنشان در مسیر درست و mount کردن گوگل درایو است. کتابخانه‌های موردنیاز در requirements.txt آورده شده‌اند و با دستور

pip install -r /path/to/requirements.txt

نصب می‌شوند. لازم است مسیر قرار گرفتن کروم درایور در کد وارد شود.

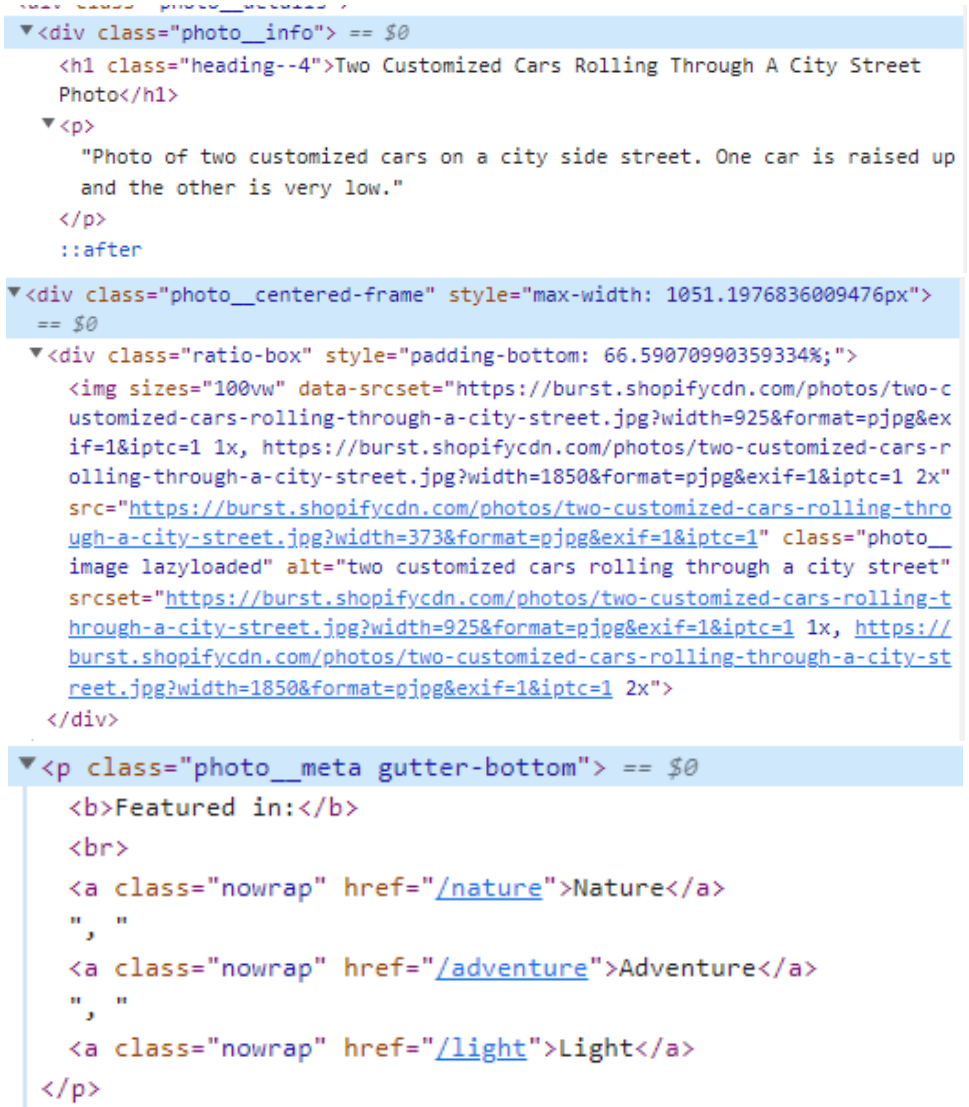
شرح کلی کد: برای crawl کردن از ابزار selenium استفاده کردیم. صفحات چندین collection از این سایت را crawl می‌کنیم. هر صفحه دارای حداکثر ۵۰ تصویر است که با زدن بر روی آن صفحه مخصوص آن تصویر به همراه عنوان و کپشن و تگ‌های آن نمایان می‌شود. برای سرعت بیشتر صفحات مربوط به ده تصویر را به صورت موازی و با multiprocessing باز می‌کنیم و اطلاعات آن‌ها را جمع آوری کرده، سپس داده های مناسب را به فایل json دیتاست اضافه می‌کنیم. در پایان با اجرای تابع download_images به کمک img2dataset تمام تصاویر موجود در فایل json دیتاست را ذخیره می‌کنیم.

به بررسی توابع موجود در کد می‌پردازیم:

تابع get_image_caption_pair: برای دسترسی به صفحه یکی از داده‌ها از مرورگر کروم به عنوان webdriver استفاده می‌کنیم سپس المان‌های داخل html صفحه را بررسی کرده و از طریق id، اسم کلاس‌ها و تگ‌ها به عنوان و توضیح تصویر و همچنین تگ‌های این تصویر دسترسی پیدا می‌کنیم. به این دلیل که به ترتیب تصاویر collectionهای متفاوت را پیدا می‌کنیم، ممکن است این تصویر قبلا در کالکشن دیگری بوده باشد و داده‌های آن ذخیره شده باشد. به همین دلیل تگ‌های آن را چک می‌کنیم تا اگر یکی از تگ‌های آن تکراری بود و در کالکشن‌های جستجو شده وجود داشت، داده آن را برنگردانیم. همچنین اگر صفحه داده بارگذاری نشود تابع لینک صفحه را برمیگرداند و در ادامه این لینک به unsuccessful_page_loads اضافه می‌شود.

```
def get_image_caption_pair(args):
    global searched_categories
    page_url = args
    driver = webdriver.Chrome('chromedriver',options=chrome_options)
    driver.get(page_url)
    try:
      info = WebDriverWait(driver, 30).until(EC.presence_of_element_located((By.CLASS_
NAME, "photo__info")))
    except:
      print(f"Couldn't get page with url: {page_url}. will add it to the unsuccessful
page loads...")
      return page_url
    title = info.find_element(by=By.TAG_NAME, value="h1").text
    caption = info.find_element(by=By.TAG_NAME, value="p").text
    tags = driver.find_elements(by=By.CLASS_NAME, value="photo__meta")[1].find_element
s(by=By.TAG_NAME, value="a")
```

```
    for tag in tags:
      if tag.text in searched_categories:
        return None
    image_holder = driver.find_element(by=By.CLASS_NAME, value="photo__centered-
frame")
    image_url = image_holder.find_element(by=By.TAG_NAME, value = "img").get_attribute
("src")
    main_url = re.search('[^\?]*', image_url).group()
    driver.quit()
    return {"title": title, "image_url": main_url, "caption": caption}
```



شکل ۳- المان‌های html بررسی شده برای گرفتن داده‌های مربوط به یک تصویر در سایت burst

تابع save_to_json_file: ابتدا فایل json دیتاست را می‌خوانیم. سپس لیست داده‌های جدید (همان ورودی تابع) را به فایل اضافه می‌کنیم.

تابع unsuccessful_page_loads: یک لینک را به فایل unsuccessful_page_loads.txt اضافه می‌کند.

تابع download_images: ابتدا فایل json را می‌خوانیم. از پکیج img2dataset استفاده می‌کنیم. یک فایل urls.txt از تمام image_urlهای درون فایل json می‌سازیم. این فایل را به عنوان ورودی به متد download می‌دهیم و با ساختاری که در بخش ۱-۱ توضیح داده شد تمامی تصاویر را دانلود می‌کند. دیگر آرگومان‌هایی که به این تابع داده‌ایم، برای تنظیم اندازه تصویر (ارتفاع/عرض تصویر هر کدام که کوتاهتر است را در صورتی که بزرگتر از ۸۰۰ باشد به ۸۰۰ با ثابت نگه داشتن ratio ابعاد تصویر تغییر می‌دهیم تا از حجم تصویر کاسته شود.)، تغییر تعداد دفعات تلاش برای دانلود یک تصویر به ۲ است.

```
#the "raw" folder along with dataset.json in it must exist in order for this to work:
def download_images():
  global dataset_path

  if os.path.exists(f"{dataset_path}/images"):
    print(f"the folder {dataset_path}/images exists. removing it...")
    shutil.rmtree(f"{dataset_path}/images")

  try:
    os.mkdir(os.path.join(dataset_path, "images"))
  except OSError as e:
    if e.errno != errno.EEXIST:
      raise

  with open(f'{dataset_path}/dataset.json', 'r', encoding='utf-8') as f:
    try:
      data = json.load(f)
    except ValueError:
      data = []
  with open("urls.txt", "w") as f:
    f.write("\n".join([d["image_url"] for d in data]))
  download(
          url_list="urls.txt",
          output_folder= dataset_path+"/images",
          resize_mode="keep_ratio",
          image_size=800,
          resize_only_if_bigger=True,
          retries = 2
        )
```

تابع handle_image_caption_pair: داده‌های multiprocess شده (خروجی تابع get_image_caption_pair) را گرفته و آن‌هایی که صفحه‌شان load نشده را در unsuccessful_page_loads گذاشته و دیتاهای دارای دیکشنری را با یک id جدید به لیست اضافه می‌کند.

تابع crawl_with_categories (تابع ساخت دیتاست): در این تابع کالکشن‌هایی که قرار است جستجو شوند قرار دارد (categories). کالکشن‌های اول اغلب تعداد داده بیشتری از کالکشن‌های بعدی دارند. دلیل این ترتیب این است که تگ‌های تصاویر بیشتری با تعداد کالکشن‌های سرچ شده کمتری چک شوند و سرعت بالاتر رود (مثلا

تصاویر کالکشن اول که شامل حدود ۳۰۰۰ تصویر است تگ‌هایش با هیچ کالکشنی چک نمیشوند، تگ‌های تصاویر کالکشن دوم تنها با نام کالکشن اول و چک میشود و همینطور تا آخرین کالکشن ادامه پیدا می‌کند.). سپس فولدر raw با فایل‌های dataset.json و unsuccessful_page_loads.txt جدید ساخته می‌شود. سپس به ازای هر کالکشن، با شروع از پیج اول و id صفر، صفحه آن گرفته می‌شود و با پیدا کردن لینک تصاویر آن صفحه به کمک المان‌های html آن، لینک‌ها به batchهای ۱۰ تایی تقسیم شده و برای هر batch به کمک multiprocessing، تابع get_image_caption_pair به طور موازی اجرا می‌شود و داده‌ها در لیست pages_data ذخیره می‌شوند. اگر این صفحه از کالکشن آخرین صفحه باشد،دکمه Next در قسمت pagination آن وجود ندارد. در این صورت از حلقه pageها خارج شده و این کالکشن را به کالکشن‌های سرچ شده اضافه کرده به سراغ کالکشن بعدی می‌رویم. در صورتی که متغیر گلوبال image_format به "file" ست شده باشد در همین تابع تصاویر هم دانلود می‌شود. می‌توان تابع دانلود تصویر را جداگانه بعد از ساخت dataset.json صدا زد.

```
def crawl_with_categories():
    global dataset_path
    global searched_categories
    driver = webdriver.Chrome('chromedriver',options=chrome_options)


    categories = ["Nature", "Seasons", "Travel", "City", "Food", "Pets", "Animal", "Pr
oduct", "Business", "Education",
                  "People", "Children", "Accessories", "Medical", "Transportation"]

    #create a "raw" folder in the folder specified by user
    path = os.path.join(dataset_path, "raw")
    dataset_path = path
    if os.path.exists(f"{dataset_path}/dataset.json"):
        print(f"the file {dataset_path}/dataset.json exists. removing it...")
        os.remove(f"{dataset_path}/dataset.json")

    try:
        os.mkdir(path)
    except OSError as e:
        if e.errno != errno.EEXIST:
            raise

    if os.path.exists(f"{dataset_path}/unsuccessful_page_loads.txt"):
        print(f"the file {dataset_path}/unsuccessful_page_loads.txt exists. removing i
t...")
        os.remove(f"{dataset_path}/unsuccessful_page_loads.txt")

    #creating dataset.json and unsuccessful_page_loads.txt files
    open(f"{dataset_path}/dataset.json",'w+').close()
    open(f"{dataset_path}/unsuccessful_page_loads.txt",'w+').close()

    pages_data = []
```

```
    searched_categories = []
    last_id = 0
    page_first = 1
    for category in categories:
      for page in range(page_first,200):
          print(f"Getting data from page {page} : https://burst.shopify.com/{category}
?page={page}")
          driver.get(f"https://burst.shopify.com/{category}?page={page}")
          main = driver.find_element(by=By.ID, value="Main")
          grid = main.find_element(by=By.TAG_NAME, value="section").find_element(by=By
.CLASS_NAME, value="js-masonry-grid")
          links = [element.get_attribute("href") for element in grid.find_elements(by
=By.TAG_NAME, value="a")]
          minibatch_length = 10
          minibatch = int(len(links) / minibatch_length)
          for iter in tqdm(range(minibatch)):
              with Pool() as p:
                  data = p.map(get_image_caption_pair, links[iter*minibatch_length:ite
r*minibatch_length + minibatch_length])
              data, last_id = handle_image_caption_pair(data, last_id)
              pages_data= pages_data + data
          remainingdata = len(links)-minibatch*minibatch_length
          if remainingdata != 0:
              with Pool() as p:
                  data = p.map(get_image_caption_pair, links[minibatch*minibatch_lengt
h:])
              data, last_id = handle_image_caption_pair(data, last_id)
              pages_data= pages_data + data
          save_to_json_file(pages_data)
          pages_data = []
          pagination = main.find_element(by=By.CLASS_NAME, value="pagination")
          if len(pagination.find_elements(by=By.LINK_TEXT, value="Next ›")) == 0:
            break #this category's pages ended
      searched_categories.append(category)
      page_first = 1
    driver.quit()
    print("================================================")
    print("json file is ready!")

    if image_format == "file":
        print("Beginning to save the images in"+ dataset_path + '/images.')
        download_images()
```

۳-۱- نحوه اجرای کد ساخت دیتاست

برای ساخت فایل دیتاست در dataset_path مسیری که در آن فولدر raw قرار می‌گیرد را قرار دهید. در صورتی که میخواهید بعد ساخت دیتاست، تصاویر دانلود شوند image_format را به "file" تغییر دهید. سپس تابع crawl_with_categories را اجرا کنید:

```
###The path where dataset folder is created on
dataset_path = "/content/drive/MyDrive"
#### change this to "file" if you want to download and save the images loc
ally after dataset.json is made.
# you can also call download_images() for this purpose.
image_format = "url"

searched_categories = []
crawl_with_categories()
```

در صورتی که فایل dataset.json در فولدر raw در مسیر داده شده به dataset_path وجود دارد، می‌توانید برای ذخیره تصاویر آن download_images را صدا بزنید.

نمونه‌ای از خروجی هنگام جمع آوری دیتاست:

```
the file /content/drive/MyDrive/raw/dataset.json exists. removing it...
the file /content/drive/MyDrive/raw/unsuccessful_page_loads.txt exists. removing it...
Getting data from page 1 : https://burst.shopify.com/Nature?page=1
100%|██████████| 5/5 [02:05<00:00, 25.11s/it]
Getting data from page 2 : https://burst.shopify.com/Nature?page=2
100%|██████████| 5/5 [01:58<00:00, 23.67s/it]
Getting data from page 3 : https://burst.shopify.com/Nature?page=3
100%|██████████| 5/5 [02:06<00:00, 25.22s/it]
Getting data from page 4 : https://burst.shopify.com/Nature?page=4
100%|██████████| 5/5 [02:02<00:00, 24.48s/it]
Getting data from page 5 : https://burst.shopify.com/Nature?page=5
 80%|████████  | 4/5 [01:35<00:24, 24.18s/it]Couldn't get page with url: https://burst.shopify.com/photos/sandy-beach-by-rocky-cliffs?c=nature. will add it to the unsuccessful page loads...
100%|██████████| 5/5 [02:19<00:00, 27.95s/it]
Getting data from page 6 : https://burst.shopify.com/Nature?page=6
100%|██████████| 5/5 [02:04<00:00, 24.91s/it]
Getting data from page 7 : https://burst.shopify.com/Nature?page=7
 80%|████████  | 4/5 [01:41<00:26, 26.17s/it]Couldn't get page with url: https://burst.shopify.com/photos/south-african-shore?c=nature. will add it to the unsuccessful page loads...
100%|██████████| 5/5 [02:17<00:00, 27.45s/it]
Getting data from page 8 : https://burst.shopify.com/Nature?page=8
100%|██████████| 5/5 [01:55<00:00, 23.09s/it]
Getting data from page 9 : https://burst.shopify.com/Nature?page=9
100%|██████████| 5/5 [02:03<00:00, 24.68s/it]
Getting data from page 10 : https://burst.shopify.com/Nature?page=10
```

۲- پیش‌پردازش داده‌ها

به جهت اینکه آموزش مدل بهتر انجام شود، لازم است ابتدا روی داده‌ها پیش‌پردازش‌هایی انجام دهیم. در اینجا به بررسی پردازش‌های انجام‌شده می‌پردازیم:

sent_broken: با استفاده از متد sent_tokenize موجود در کتابخانه nltk هر کپشن را به جملات آن می‌شکنیم.

```
def sent_broken(dataset):
    captions = [d['caption'] for d in dataset]
    sentences = [sent_tokenize(caption) for caption in captions]

    new_data = [None] * len(dataset)
    idx = 0
    for d, s in zip(dataset, sentences):
        new_data[idx] = {'id': d['id'],  'title':d['title'], 'image_url':
d['image_url'] , 'caption': s}
        idx += 1
```

```
    return new_data
```

word_broken: با استفاده از متد word_tokenize موجود در کتابخانه nltk هر کپشن را به کلمات آن می‌شکنیم.

```
def word_broken(dataset):
    captions = [d['caption'] for d in dataset]
    words = [word_tokenize(caption) for caption in captions]

    new_data = [None] * len(dataset)
    idx = 0
    for d, w in zip(dataset, words):
        new_data[idx] = {'id': d['id'],  'title':d['title'], 'image_url':
d['image_url'] , 'caption': w}
        idx += 1

    return new_data
```

clean_dataset: در ابتدا تمامی علائم نگارشی به جز نقطه و همچنین اعداد را از کپشن‌ها حذف می‌کنیم. سپس با استفاده از WordPunctTokenizer توکن‌ها را بدست می‌آوریم (به جهت حذف نشدن نقطه‌ها از این tokenizer استفاده شد). در مرحله آخر تمامی حروف به lower case تبدیل شدند و آنها را با کاراکتر فاصله به هم متصل کردیم. داده‌های بدست آمده را در یک فایل با فرمت json ذخیره کردیم.

```
def clean_dataset(dataset):
    porter = PorterStemmer()

    tokenizer = WordPunctTokenizer()
    cleaned_data = [ ]
    for d in dataset:
        caption = d['caption']

        # remove punctutaions except '.'
        caption = re.sub(r'[^\w\s\.]', '', caption)

        # remove numbers
        caption = re.sub(r'[\d]', '', caption)

        # split by word
        caption = tokenizer.tokenize(caption)

        # convert to lower case
        caption = [word.lower() for word in caption]

        cleaned_data.append(' '.join(caption))

    new_data = [None] * len(dataset)
```

```
    idx = 0
    for d, c in zip(dataset, cleaned_data):
        new_data[idx] = {'id': d['id'],  'title':d['title'], 'image_url':
d['image_url'] , 'caption': c}
        idx += 1

    return new_data
```

چند نمونه از کپشن‌ها قبل و بعد از اعمال پیش‌پردازش:

```
original:
Little girl rides on father's shoulders on the grass.
cleaned:
little girl rides on fathers shoulders on the grass .

original:
A lighthouse sits on the edge of a rich green cliffside surrounded by
daisies.
cleaned:
a lighthouse sits on the edge of a rich green cliffside surrounded by
daisies .

original:
Smiling woman working. Her blazer and laptop are in matching colors.
Smiling toward her coworkers.
cleaned:
smiling woman working . her blazer and laptop are in matching colors .
smiling toward her coworkers .

original:
Business women sits in front of an open laptop as she writes in a
notebook. There is a camera lens on the table as well as a houseplant.
cleaned:
business women sits in front of an open laptop as she writes in a notebook
. there is a camera lens on the table as well as a houseplant .

original:
A tiny island is illuminated by street lights and buildings around it's
edge, set in the middle of a large lake.
cleaned:
a tiny island is illuminated by street lights and buildings around its
edge set in the middle of a large lake .

original:
From up here, the city looks almost like one being, its jagged edges
piercing the pink sky. It's the magical moment when the sunlight is almost
gone, but the city lights have not turned on just yet.
cleaned:
from up here the city looks almost like one being its jagged edges
piercing the pink sky . its the magical moment when the sunlight is almost
gone but the city lights have not turned on just yet .

original:
Two rock climbers stand together at the foot of a wall, planning their
climbing route.
cleaned:
two rock climbers stand together at the foot of a wall planning their
climbing route .
```

```
original:
A cloud of mist rises above a rocky water fall, a picturesque autumn scene
with red brick buildings and trees in different shades of green and brown.
cleaned:
a cloud of mist rises above a rocky water fall a picturesque autumn scene
with red brick buildings and trees in different shades of green and brown
.

original:
White bridge over a canal that flows through the bottom of the frame.
There is a orange building that is right on the water with trees growing
out of a balcony.
cleaned:
white bridge over a canal that flows through the bottom of the frame .
there is a orange building that is right on the water with trees growing
out of a balcony .

original:
Cool hotdog enamel pin on a man's jean jacket.
cleaned:
cool hotdog enamel pin on a mans jean jacket .
```

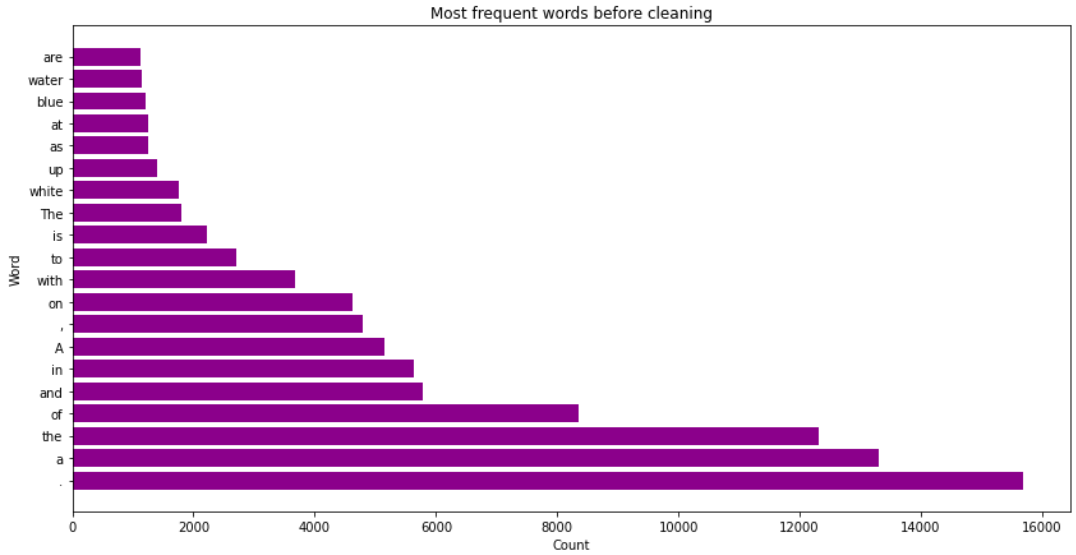
۳- آمار و ارقام

در جدول زیر آمار دیتاست قبل و بعد از انجام پیش‌پردازش ذکر شده است.

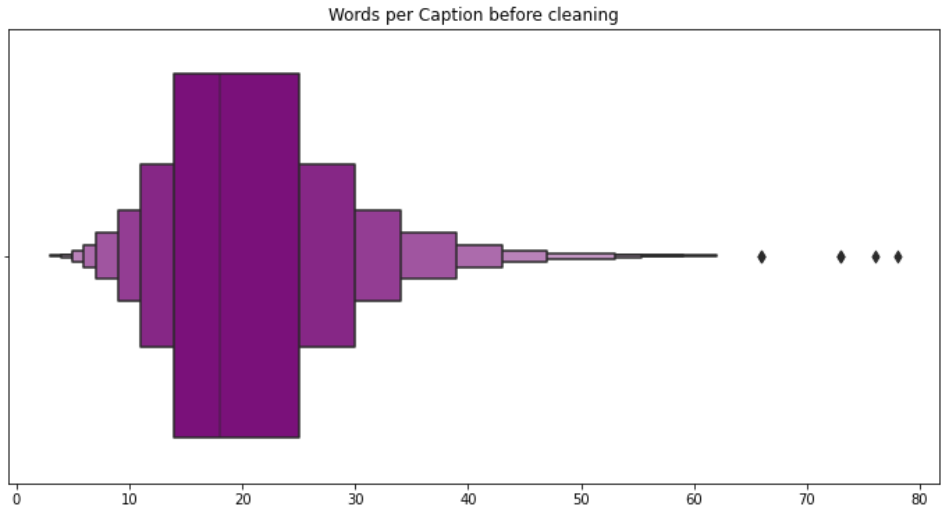
	قبل از پیش‌پردازش	پس از پیش‌پردازش
تعداد نمونه‌های موجود در دیتاست	12602	12602
تعداد جملات	16664	16404
تعداد واژگان	249682	242598
تعداد کلمات منحصر به فرد	12135	10333

۳-۱- قبل از پیش‌پردازش

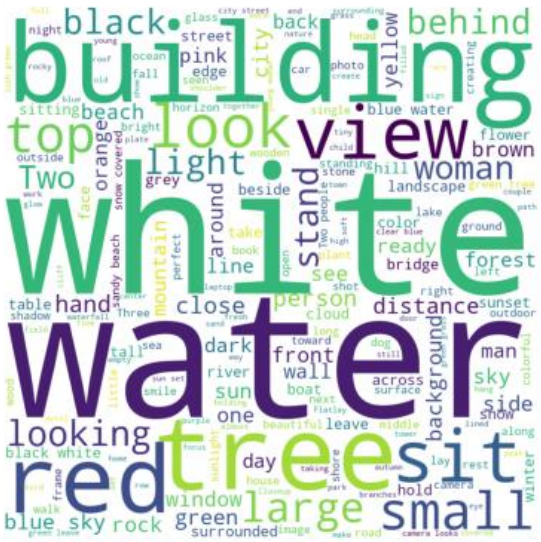
نمودار ۲۰ واژه پرتکرار بر حسب تعداد تکرار مطابق زیر است:



نمودار توزیع تعداد کلمات موجود در هر کپشن مطابق زیر است:

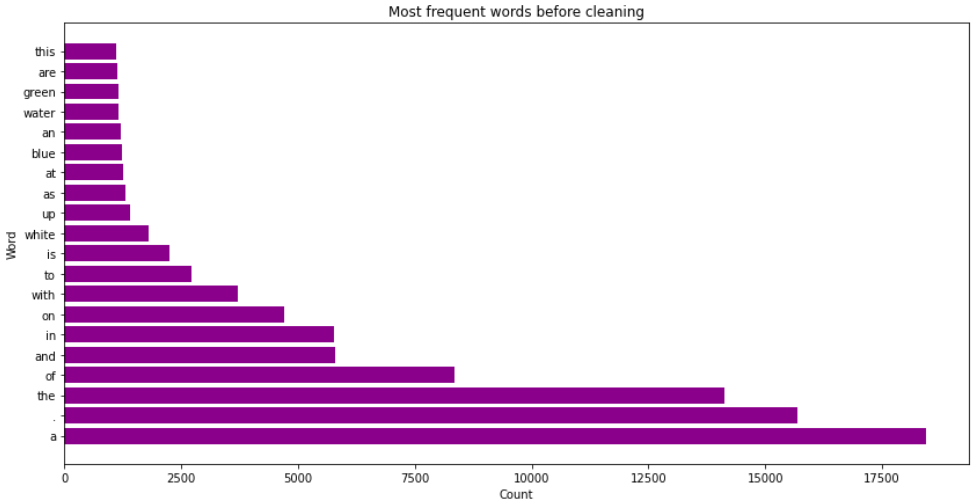


Word cloud کلمات موجود در دیتاست به صورت زیر است:

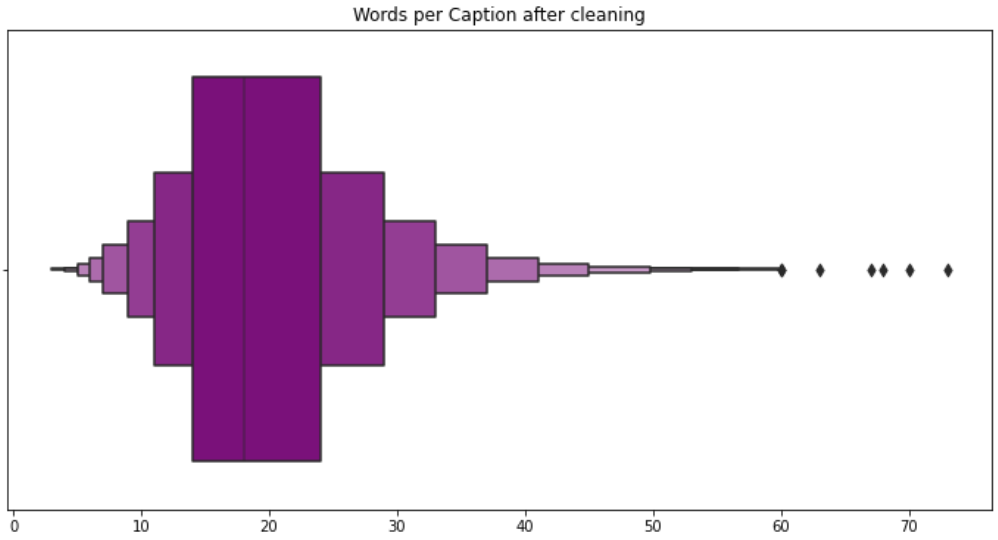


۲-۳- بعد از پیش‌پردازش

نمودار ۲۰ واژه پرتکرار بر حسب تعداد تکرار مطابق زیر است:



نمودار توزیع تعداد کلمات موجود در هر کپشن مطابق زیر است:



Word cloud کلمات موجود در دیتاست به صورت زیر است: