Louis Atallah
07/22/2022

Divercity ML Engineer Intern Python Challenge

Summary

Part 1: LinkedIn web scraper

The first part of this challenge is to build a program able to web scrap LinkedIn data by itself, without interferences. To do so, I used a Jupyter Notebook script which allows me to execute only specific cells of code and is convenient for the web scraping. The code starts with all the package imports necessary to build my web scraper. I created the class LinkedInScraper to get the web links of employee profiles with their names and labels. The constructor takes two parameters, keyword and limit, the company's name and the number of profiles to get from the specified company.

It includes a search() function that takes no argument, will perform the search using a random user agent among the ones provided, and returns a list of HTMLs from Google Searches. The parse_links() method collects the profile links and returns a list of links. There is no argument to pass as parameter. The third functionality of the LinkedInScraper class is parse_people(). It parses the html for the LinkedIn profiles using regex, and returns a list of employees with their names and their labels. The regex expression [â-zA-z0-9_] will match only a single non-alphanumeric/underscore character. I used it to get the information from the HTMLs. It is from the re package, known as regex, as well.

To navigate on the web and login, I choose Chrome with its web driver for Chrome v.103. Once the web driver downloaded, I specify the incognito mode with language set to English to avoid interferences.

For the login part, I implemented a function login() as it will be reused later in the code. Login() leads us to the LinkedIn Sign in web page and pass the credentials given in input. The event is validated by clicking on the sign in button. I reached the button event by using CSS selectors after having carefully inspected the element on the web page. I could do so with the selenium library. It enables to find an element on a web page by its id or class name or title. With the class name of the element, I simulate a click which brings us to the LinkedIn feed.

```
def login():
    """

    Function to get  to the login page and sign in
    """
    driver.get("https://www.linkedin.com/")

    email = input("email/cell phone: ")
    password = input("password: ")

    username = driver.find_element_by_id("session_key")
    passw = driver.find_element_by_id("session_password")
    username.send_keys(email)
    passw.send_keys(password)

    element = driver.find_element_by_class_name("sign-in-form__submit-button")
    try :
        element.click()
    except Exception as e :
        print(e)
```

Once logged in, I pass the link of the company as url of the get() method from driver. We are now on the company Hubspot LinkedIn web page.

At this step, the LinkedIn scraping is about to start. I instantiate an element named “ls” of the LinkedInScraper() class with Hubspot as keyword and 250 as limit. I put a limit of 250 to have more profiles than what is required to perform a data cleaning on it and avoid empty rows or no data.

LinkedIn Scrapping

```
ls = LinkedinScraper(keyword="Hubspot",limit=300)
ls.search()
links = ls.parse_links()
profiles = ls.parse_people()
```

Now that I have the links, I use a for loop to explore the 250 profiles loaded and extract the location and the profile picture along the first names, last names, and labels.

To deal with the various data entries, I perform a split(“ – “) to create a list per employee including their names and labels (“profile” in the code). Accessing elements might be hazardous with some data or website events, so I implemented some try/except to deal with errors that could occur, either because of the name formatted differently or LinkedIn kicked the driver out with a warning message.

Still in the for loop, a driver request is made for each link stored. Eventually, we get the data via driver.find-element_by_xpath(path) coming from selenium. The path is the one to the CSS element.

At this step, I have been warned many times from LinkedIn due to a strange activity as the scraper profile was a new account that exceeds the number of profile views. At 52 requests in a row regarding profile views, LinkedIn terminate the sessionand could ban my account.

To avoid the ban, I processed by batch of 50 profiles. I create a temporary pandas data frame to store the data freshly pulled and concatenate it with the main data frame. The data frame has 5 columns which are: firstname, lastname, label, location, profile_picture.

I perform some data cleaning on it, such as dropping duplicated rows, and dropping rows containing Nas as firstname, lastname, or profile_picture.

If the number of rows is still superior to 200, the reduce() function returns a list of index with nan values containing int("lim") index. It takes "lim" as the only parameter.

```
def reduce(lim) :
    """
    return index with nan values
    """
    rows_with_nan = []
    for index, row in dataset.iterrows():
        is_nan_series = row.isnull()
        if is_nan_series.any():
            rows_with_nan.append(index)
    rows_with_nan = rows_with_nan[:lim]
    return rows_with_nan
```

Now that the data has been pulled from the Internet, I load it into a csv file named "data_employees.csv" (data_employees_base.csv to refer).

As I showed in the screen recording, the pictures are downloaded from the profiles and stored in a folder named "pictures" with the code below:

```
for i in range(200) :
    url_img = d.iloc[i][4]
    img_name = " ".join([str(d.iloc[i][0]),str(d.iloc[i][1])])
    urllib.request.urlretrieve(
                url_img,
                filename="./pictures/{}.png".format(img_name))
```

The code above enables to retrieve pictures in the dataset and download them with their employee names.

Part 2: Gender prediction

For the second part of the challenge, I put the libraries used for this part at the beginning of the section. Most libraries imported are TensorFlow/Keras or image related as I perform a binary classification on images.

I chose VGG16 as a Convolutional Neural Network (CNN). It is a model created in 2014 developed and trained by Google and Oxford University. It is 16 layers deep and has over 130 million of parameters. The input has the format of (224,224,3) passed as input_shape argument. Usually, it is a good shape with a RGB mode. The argument weights as imagenet indicates that the CNN uses its previous trained data.

The basic layers are not trained again, so we set their trainable attributes to false.

```
base_model = VGG16(input_shape = (224, 224, 3), # Shape of our images
include_top = False, # Leave out the last fully connected layer
weights = 'imagenet')
```

basic layers are not trained again

```
for layer in base_model.layers:
    layer.trainable = False
```

Then, I add 4 layers to the base model. A flatten layer that will flatten the output to 1 Dim, a fully connected layer with 512 hidden units and a ReLU activation, a dropout rate of 0.5, and a sigmoid layer with one node for classification output.

The model is compiled with the Adam optimizer with a learning rate of 0.0001. It is known to be efficient on image treatment. With the model.summary() call, we can see the composition of the CNN along the layers stacked with their number of parameters. The model has a total of 27 million of parameters.

To get the images from the urls, I created a method to return the image properly formatted. The open_convert() function checks that the url is complete, try to open the image with an urlib.request from the PIL package, convert it into RGB color mode, resize and reshape it as well as the model input requires, and returns the image as an array of numeric values.

```
def open_convert(url) :
    """
    return the image properly formatted
    """
    if url[0:4] == 'http' : # if url is complete
        try :
            im = Image.open(urllib.request.urlopen(url))
        except :
            print("Error on url : "+url)
            im = None
        if im != None : # if there is an image
            if im.mode != 'RGB' : # conversion into RGB colors
                im = im.convert('RGB')
            im  = im.resize((224,224), Image.Resampling.LANCZOS)  # proper size

            img =im
            img_array = img_to_array(img)
            img_array = img_array.reshape((1,) + img_array.shape)
            return img_array
```

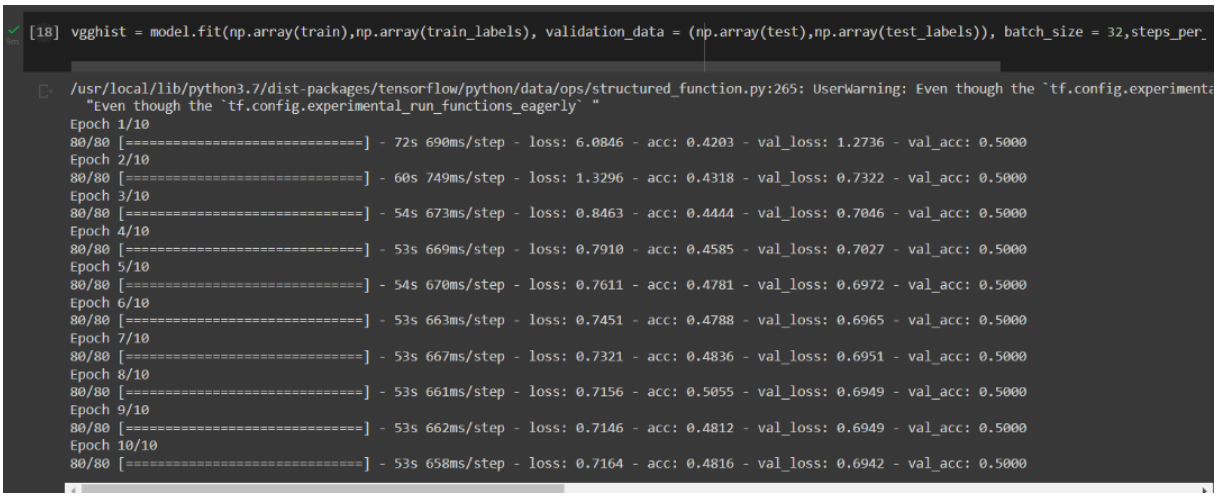
To train the model, I had to gather gender data. I went on Kaggle to take folders of male and female pictures as training and validation sets. I get the images from the folders by calling the get_image(sex,trainset,labels,step) method that will get the image along their sex. Using the glob package, I do a for loop to explore all the .jpg files in the folder. The image transformation is the same as the one in the previous method. It returns two lists: one for the images, and the second for the sex. Each element in both lists are respectively ordered. In other words, trainset[i] matches labels[i]. It takes the sex (male or female), a list of images, a list with their sex, and “step” the repository name (either training or validation).

```
def get_image(sex, trainset=train, labels = train_labels, step = 'training') :
    """
    get images from the training and validation sets
    those sets are the labelized data I gathered from Kaggle to train the model (supervised learning)
    """
    for filename in glob.glob('./'+str(step)+'/male/*.jpg'): # all .jpg files

        im=Image.open(filename)
        if im.mode != 'RGB' :
            im = im.convert('RGB')
        im  = im.resize((224,224), Image.Resampling.LANCZOS)

        img =im
        img_array = img_to_array(img)
        img_array = img_array.reshape(img_array.shape)

        if sex == 'male' :
            trainset.append(img_array)
            labels.append(1)   # male
        else :
            trainset.append(img_array)
            labels.append(0)  # female

    return trainset, labels
```

During the study, I have been restricted to a training set of 3000 rows (men+women) and a validation set of 610, evenly distributed. It gives 80% for the training and 20% dedicated to the validation test. The lack of data is due to the size of the images and the time taken to compute the training as I received a MemoryError. In order not to get stuck, I used Google Colab. It is a great way to run code online using the GPU. The GPU enabled me to process

the fitting part successfully even if I still couldn't load more data as the Colab repositories are essentially online platforms such as Google Drive.

I took a batch size of 32 on 10 epochs with 80 steps per epochs. The history result from the training displayed below:



```
[18] vgghist = model.fit(np.array(train),np.array(train_labels), validation_data = (np.array(test),np.array(test_labels)), batch_size = 32,steps_per_

/usr/local/lib/python3.7/dist-packages/tensorflow/python/data/ops/structured_function.py:265: UserWarning: Even though the `tf.config.experimenta
  "Even though the `tf.config.experimental_run_functions_eagerly` "
Epoch 1/10
80/80 [==============================] - 72s 690ms/step - loss: 6.0846 - acc: 0.4203 - val_loss: 1.2736 - val_acc: 0.5000
Epoch 2/10
80/80 [==============================] - 60s 749ms/step - loss: 1.3296 - acc: 0.4318 - val_loss: 0.7322 - val_acc: 0.5000
Epoch 3/10
80/80 [==============================] - 54s 673ms/step - loss: 0.8463 - acc: 0.4444 - val_loss: 0.7046 - val_acc: 0.5000
Epoch 4/10
80/80 [==============================] - 53s 669ms/step - loss: 0.7910 - acc: 0.4585 - val_loss: 0.7027 - val_acc: 0.5000
Epoch 5/10
80/80 [==============================] - 54s 670ms/step - loss: 0.7611 - acc: 0.4781 - val_loss: 0.6972 - val_acc: 0.5000
Epoch 6/10
80/80 [==============================] - 53s 663ms/step - loss: 0.7451 - acc: 0.4788 - val_loss: 0.6965 - val_acc: 0.5000
Epoch 7/10
80/80 [==============================] - 53s 667ms/step - loss: 0.7321 - acc: 0.4836 - val_loss: 0.6951 - val_acc: 0.5000
Epoch 8/10
80/80 [==============================] - 53s 661ms/step - loss: 0.7156 - acc: 0.5055 - val_loss: 0.6949 - val_acc: 0.5000
Epoch 9/10
80/80 [==============================] - 53s 662ms/step - loss: 0.7146 - acc: 0.4812 - val_loss: 0.6949 - val_acc: 0.5000
Epoch 10/10
80/80 [==============================] - 53s 658ms/step - loss: 0.7164 - acc: 0.4816 - val_loss: 0.6942 - val_acc: 0.5000
```

The accuracy is not high as it is only 0.5 on the validation set

This is due to the lack of data for the fitting part. As Colab run online, it was a challenge to feed and train the model. Colab processed the code faster, but as the repository is online it takes time and space in addition to that my Wi-Fi is very busy.

I load the model computed from script ran on Colab by loading it to a json model from the model_from_json library with its weights too.

Now that the model is trained, we can perform predictions on the dataset. I applied the predict attribute from the model to each image of the dataset. It turns out that 106 men and 94 women were predicted out of the 200 random Hubspot employees on LinkedIn.

106 men and 94 women predicted

```
pred.count(1), pred.count(0)
```

```
(106, 94)
```

I merge the result with the base data frame (produced from Part. 1), and replace digits by categorical string values as below:

```
d['pred'] = pred
```

```
d['pred'] = d['pred'].replace(0,"female")
d['pred'] = d['pred'].replace(1,"male")
```

The final csv is stored as “data_employees_gender.csv”.

	firstname	lastname	label	location	profile_picture	pred
43	Rachel	Leist	Senior Director of Marketing at HubSpot	Boston, Massachusetts, United States	https://media-exp2.licdn.com/dms/image/C4E03AQ...	male
174	Nathan	Hodge	Corporate Account Executive	Portland, Maine, United States	https://media-exp1.licdn.com/dms/image/C5603AQ...	female
23	Mark	Roberge	ark Roberge - Managing Director - Stage 2 Capital	Cambridge, Massachusetts, United States	https://media-exp2.licdn.com/dms/image/C4D03AQ...	female
3	Brian	Halligan	rian Halligan - Executive Chairperson	Cambridge, Massachusetts, United States	https://media-exp2.licdn.com/dms/image/C4E03AQ...	male
79	Ryan	Batter	Director Corporate Strategy	Greater Boston	https://media-exp2.licdn.com/dms/image/C4E03AQ...	male
103	John	Eldridge	Corporate Growth Representative	Boston, Massachusetts, United States	https://media-exp1.licdn.com/dms/image/C5603AQ...	female
19	Henry	Wu	NaN	Washington, District of Columbia, United States	https://media-exp2.licdn.com/dms/image/C5103AQ...	male
109	Mali	R.	Channel Account Manager	Milwaukee, Wisconsin, United States	https://media-exp1.licdn.com/dms/image/C4D03AQ...	female
184	Kevin	Ackhurst	NaN	Greater Boston	https://media-exp1.licdn.com/dms/image/C4D03AQ...	male
26	Jorie	Munroe	orie Munroe - Senior Inbound Professor	San Francisco Bay Area	https://media-exp2.licdn.com/dms/image/C5603AQ...	female
51	Clarina	He	Business Development Representative	Suffolk County, Massachusetts, United States	https://media-exp2.licdn.com/dms/image/C5603AQ...	female
24	Brandon	Greer	randon Greer - Head of HubSpot Ventures	Cambridge, Massachusetts, United States	https://media-exp2.licdn.com/dms/image/C4E03AQ...	female
192	Magdaline	Small	HubSpot for Startups	Nashville, Tennessee, United States	https://media-exp1.licdn.com/dms/image/C4D03AQ...	male
92	Diehl	Longstaff	Account Executive - Small Business	Los Angeles Metropolitan Area	https://media-exp1.licdn.com/dms/image/C4D03AQ...	male
112	Head	of	NaN	Greater Boston	https://media-exp1.licdn.com/dms/image/C4E03AQ...	female
185	Daniel	Dias	Account Executive - Mid Market - HubSpot	Cape Coral Metropolitan Area	https://media-exp1.licdn.com/dms/image/C4D03AQ...	male
165	Giles	Ober	NaN	Cambridge, Massachusetts, United States	https://media-exp1.licdn.com/dms/image/C5603AQ...	male

Conclusion

To conclude this study, coding a web crawler works well from Chrome with Chrome driver, it pulls properly the data and doesn’t need to be assisted.

The second part, about the convolutional neural network classifier to predict the sex, gathers the VGG16 model with its additional layers and perform an accuracy of 0.5 on the test set. It gives as output 106 men and 94 women.

There were many challenges inside this assignment in terms of tool capacity, for example, the limited number of profile views on LinkedIn. This leads to think about a way to avoid exceptions by batch processing the requests. For the classifier, it was more complicated as far as the memory and the Internet connection were concerned. In fact, the images are large, the CNN is too heavy

(more than 16 layers and 27 million param.) to be supported by my engine, and the Wi-Fi connection was not sufficient to upload MB of files.

To go further, I would improve the classifier by fitting on more data to have a better accuracy. I would also apply hyper tuning to the model by implementing a grid search to determine which learning rates, optimizers (Adam, Adamax, RMSprop…), and loss functions that would give the best accuracy.

I would extract the vectors generated by the model from the profile pictures by removing some layers on the model. Perform a proper train. Once I have all the images vectorized in the same dimensional space, I would perform an ensemble modeling technique by adding a k-means algorithm to gather the vectors into 2 clusters (male and female).