Assignment 1 Big Data Engineering

Kinjo Somopawiro 500802522 BDS1

25 October 2021

Table of contents

Contents

Summary

In this report I will be writing about the many findings during the process of scraping reviews from websites and using a various amount of machine learning techniques to understand the real meaning of a written review and the entire process bringing us to that point.

Introduction

The use data and machine learning are some of the most talked about topics when it comes to the IT world. Yet still surrounded mystery by the public in terms of how it exactly works. This document will explain the uses and process of collecting data and using it for machine learning.

methods

Acquiring data

A popular method that I used to get reviews from websites is called “Web Scraping”. Web Scraping is the process of collecting data from a website in an automated way. An alternative name is also called web data extraction. web scraping is used for things like price monitoring, price intelligence, news monitoring and market research among many other activities that involve data from websites.

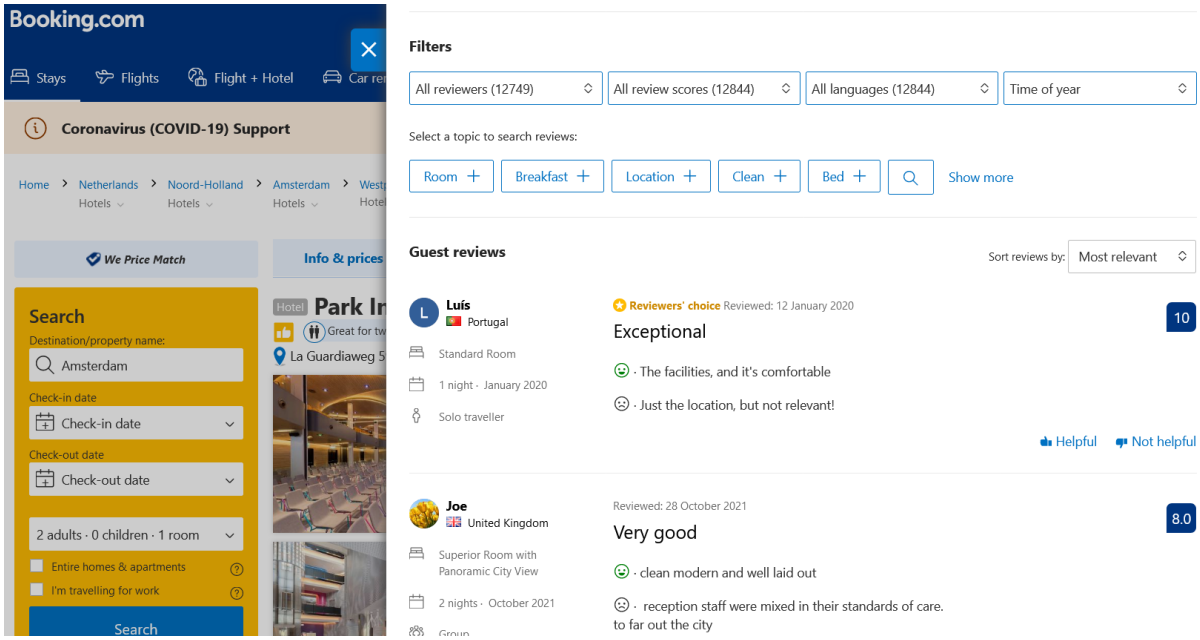
For the web scraping part, I made use of a “WebDriver”. A WebDriver uses browser automation APIs provided by the web browser vendors to control a web browser and run tests. With this it can replicate the action of a real user within web browser.

The WebDriver that was used for this report is called “Selenium” in combination with the Firefox browser.

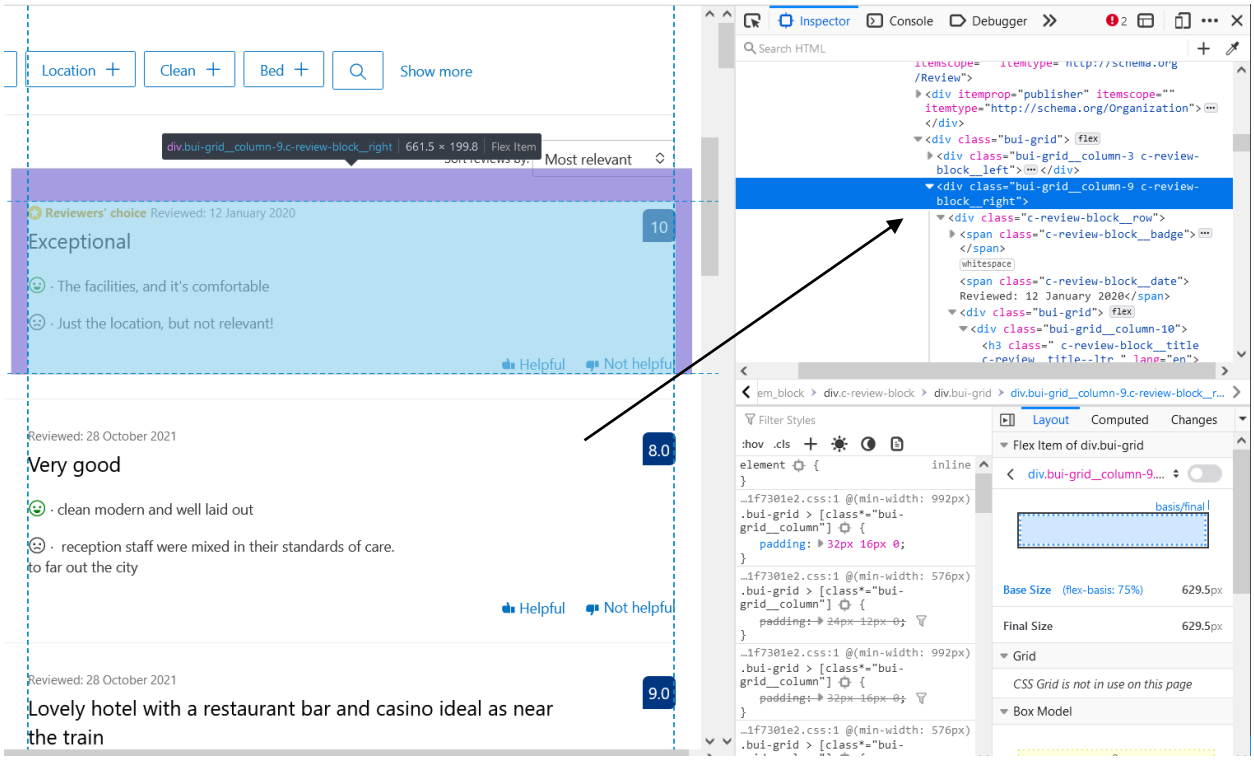
In the Web scraper code

With the web driver we are able to reach the desired website trough the given link.

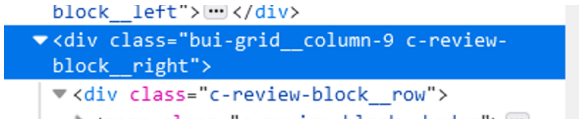
```
# going to the website with the webdriver
driver = webdriver.Firefox(executable_path="geckodriver.exe")
driver.get('https://www.booking.com/hotel/nl/park-inn-by-radisson-amsterdam-
city-west.en-gb.html?aid=304142;label=gen173nr-
1FCBcoggI46AdIM1gEaKkBiAEBmAEJuAEXyAEP2AEB6AEB-
AECiAIBqAIDuAKzldqLBsACAdICJDNhY2UwMjFkLWQ4MTYtNGYyMS1iMjk2LTIxODIwNDc1ZTY4Odg
CBeACAQ;sid=7ba796949fabe9953261480dff3918ad;dest_id=-
2140479;dest_type=city;dist=0;group_adults=2;group_children=0;hapos=0;hpos=0;n
o_rooms=1;req_adults=2;req_children=0;room1=A%2CA;sb_price_type=total;sr_order
=popularity;srepoch=1635179469;srpvid=17c67425457b00b0;type=total;ucfs=1;sig=v
1SlkxZCZN&#tab-reviews')
```



Once the web driver has reached the desired website, we want to start looking for the reviews that we are going to collect. Every element within a webpage has an ID that it can use to locate our data. With the inspecting tool that is built in the web browser we can look for the ID that fits with the desired data.



We can give to ID of the html to the driver so that it can find the data we need and start collecting




```
# find the reviews by html/xpath ID
reviews = driver.find_elements_by_xpath("//*[@class='bui-grid__column-9 c-review-block__right']")
```

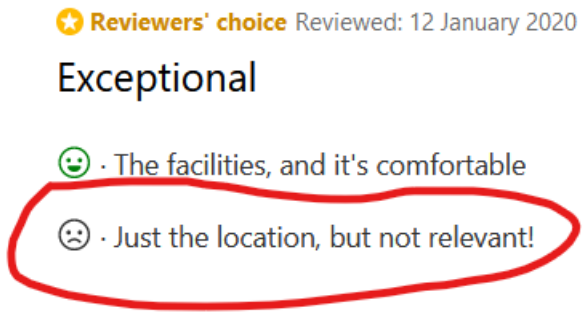
Within the code when we go a little furter into the ID so that we can collect more precise data we need. For example the title with their corresponding title ID within the html. We do this with the review text(`scraped_dubbel_review`) and the given score.kj

```
    for i in range(len(reviews)):
 title = reviews[i].find_element_by_xpath(".//div[@class='bui-grid__column-10']").text
 scraped_dubbel_review = reviews[i].find_element_by_xpath(".//div[@class='c-review']").text.replace("\n", " ").replace("Liked", " ")
 score = reviews[i].find_element_by_xpath(".//div[@class='bui-review-score__badge']").text
```

After this we clean the text a little and separate the negative and positive part of the review in separate rows that we will later use for our classification training sets.

```
    # split review into negative ans positive part
     scraped_dubbel_review = scraped_dubbel_review.split("Disliked")

     # determine the sentiment of the review
     for i in range(len(scraped_dubbel_review)):
         review = scraped_dubbel_review[i]
         if((scraped_dubbel_review[i]!="") and (i==0)):
             scrapedReviews.append([title,review,score, 1,"booking"])
         elif((scraped_dubbel_review[i]!="") and (i==1)):
             scrapedReviews.append([title,review,score, 0,"booking"])
```

In the screenshots below you can see that we gave reviews in the negative column a 0 and reviews in the positive column a 1 so that we can differentiate them and later on use to it to tell our algorithms how a ‘good’ and ‘bad’ review looks like.



21	very good	· Air-condition	8.0	1	booking
22	Exceptional	· Just the location, but not relevant!	10.0	0	booking
23	Very good	Breakfast and room quality	8.0	0	booking

after all the data has been collected, we will convert it into a CSV data file that we can use to store our data and use it later again.

```
# put scraped review into a dataframe
scrapedReviewsDF = pd.DataFrame(scrapedReviews, columns=['title',
'review', 'rating','sentiment','source'])
driver.quit()
scrapedReviewsDF.head(60)
print( 'Ready scraping ....')
scrapedReviewsDF.to_csv("booking_reviews.csv", sep=',',index= False)
```

Storing and calling the collected data

As mentioned before, we have stored our web scraped data in a “CSV” file. CSV stands for “Comma Separated Values” which as the name applies stores the data and can differentiate it by looking for the commas.

A screenshot within the CSV file we used to store our data. It uses commas between data entries so that it can later read them back again

```
title,review,rating,sentiment,source
Exceptional,"   · The facilities, and it's comfortable ",10,1,booking
Exceptional,"  · Just the location, but not relevant!",10,0,booking
Lovely hotel with a restaurant bar and casino ideal as near the train,"   · good locatior
Nice basic hotel,"   · Nice view, room was okay ",8.0,1,booking
Nice basic hotel,  · Hotel people were somehow very slow,8.0,0,booking
Good,  · bed was not king size. it was 2 bed stacked I came this summer and the burger DI
Brilliant few days stay except for pillow issue,   · Maid was good .staff were friendly a
Brilliant few days stay except for pillow issue,  · 2 pillows each and better explanatior
Good,"   · Staff were nice, nicr location very accessible to the coty4 ",7.0,1,booking
Good,  · The food in the restaurant we had for lunch was inedible. Both the beef burger a
we enjoyed very much,   · the location was so beautiful ,8.0,1,booking
we enjoyed very much,  · nothing,8.0,0,booking
"Perfect hotel, clean, not so expensive and close to subway station",   · The place is ar
"Perfect hotel, clean, not so expensive and close to subway station",  · Quite a long way
Pleasant,"   · the bed was amazing, the room rate was excellent near to train station sor
Very good,   · Rooms a bit too small. ,8.0,1,booking
```

After scraping from a bunch of websites we have collected multiple CSV files that contain our data. This chapter will show how the files are merged and send trough the database and how we read the files again to use for the ‘sentiment analysis’.

Sending the data to the database

```
keagle_positive_reviews_df =
pd.read_csv('Hotel_Reviews.csv',sep=',').rename(columns={"Positive_Review":"review","
Hotel_Name":"title","Reviewer_Score":"rating"})

keagle_negative_reviews_df =
pd.read_csv('Hotel_Reviews.csv',sep=',').rename(columns={"Negative_Review":"review","
Hotel_Name":"title","Reviewer_Score":"rating"})
trip_advisor_reviews_df = pd.read_csv('trip_advisor_review.csv',sep=',')
booking_reviews_df = pd.read_csv('booking_reviews.csv',sep=',')
```

In the code we are reading al our CSV files and transform them in a data frame(**a 2-dimensional labeled data structure with columns of potentially different types**). When the CSV file is called its also changes some of the column names

```
).rename(columns={"Positive_Review":"review"
```

so that it will match with the other files and merge together more smoothly.

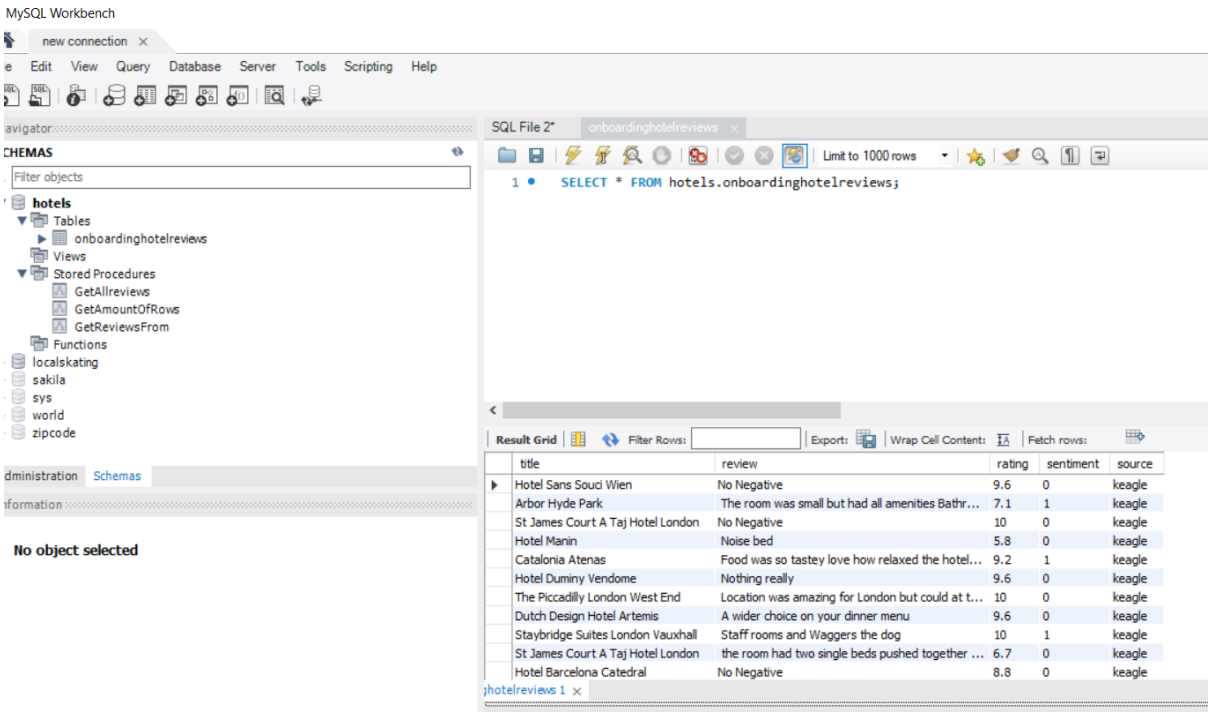
In the code below all the data frames are stacked together with `.append()` afterwards we use `.sample(frac = 1)`to shuffle the data and `.reset_index()`to reset to index numbers

```
df=keagle_positive_reviews_df.append(keagle_negative_reviews_df).append(booking_revie
ws_df).append(trip_advisor_reviews_df).sample(frac = 1).reset_index(drop=True)
```

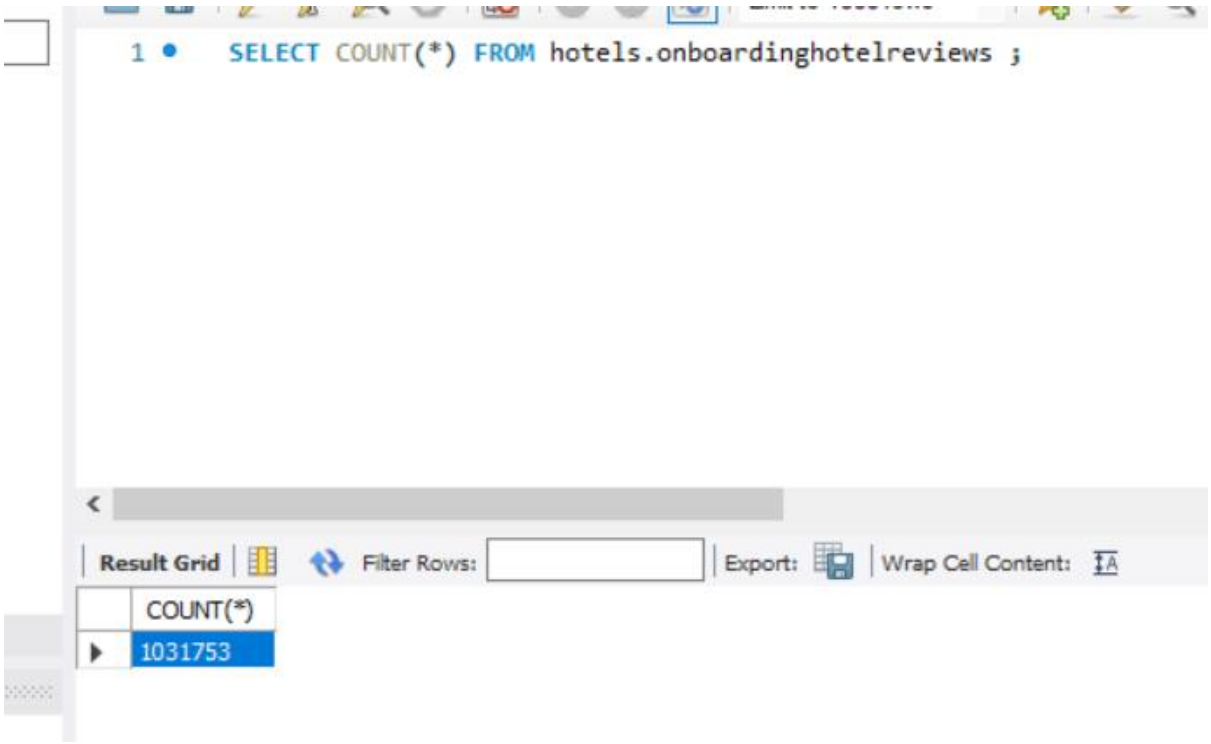
Now that alle out data has merged, a connection will be made with the database so that data can be brought over there

```
# create db first in MySQL
engine = create_engine('mysql+mysqlconnector://root:root@localhost/hotels')
df.to_sql(name='onboardinghotelreviews', con=engine, if_exists='fail',
index=False,chunksize=1000)
```

Now that the data is in the database we can create some “Stored procedures” to call the data in an efficient way by making pre created query’s.



An example of one of the created stored procedures is “GetAmountOfRows(row_amount)”. After testing a fair amount of data, it will be inevitable to come across a time problem where it just takes hours to finish a test (There over a million rows). So it helps to limit the data to a certain amount and reduce compiling time with this created stored procedure()




```
DELIMITER //
CREATE PROCEDURE GetAmountOfRows(IN row_amount INT)
BEGIN
        SELECT *  FROM hotels.onboardinghotelreviews LIMIT row_amount;
END     //
DELIMITER ;
CALL GetAmountOfRows(row_amount)
```

To read our data again we simply just create a connection again and call one of our stored procedures to load everything in python.

```
# read reviews from db in MySQL
engine =
create_engine('mysql+mysqlconnector://root:root@localhost/hotels'
)
dbConnection = engine.connect()
df = pd.read_sql("CALL GetAllreviews()", dbConnection);
dbConnection.close()
```

Training and classifying

Now that the data is stored and can be read however and whenever we need. We will start with our "Sentiment Analysis" to find the subjective state of the data. In this case we will see if it can accurately predict if a review is positive or negative based on the words used.

Making the data "machine learn ready"

Before the data is passed trough the machine learning process, it needs to be cleaned to ensure the best results and making sure that it is readable for the program itself since it can only take in 0s and 1s despite our reviews being made from words.

In this code snippet all the special characters are removed to make it easier to find the most frequently counted word. We count all the words to find out which ones are unnecessary for the sentiment analysis.

```
# remove special charachters
df['new_review'] = df.review.str.replace("[^a-zA-Z#]", " ")

# finding the most frequent words
all_words = []
for line in list(df['new_review']):
    words = line.split()
    for word in words:
        all_words.append(word.lower())


a=Counter(all_words).most_common(10)
print(a)

# output
('the', 10214), ('and', 6070), ('was', 4630), ('to', 4153), ('a', 3777),
('room', 3080), ('in', 2683), ('very', 2622), ('no', 2382), ('staff',
2300)]
```

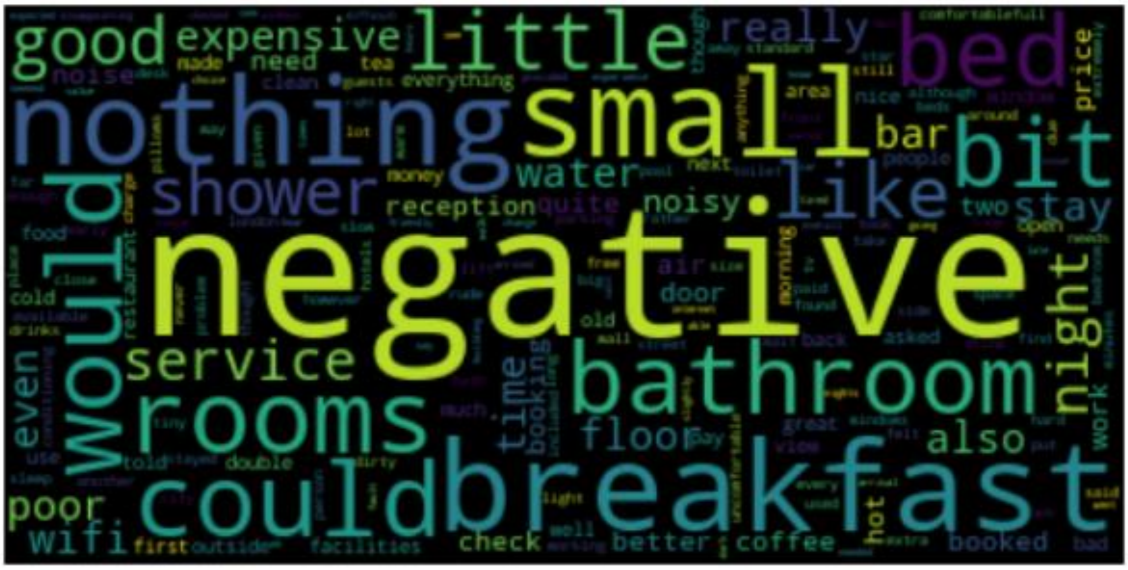
In this code snippet we apply tokenization to combine similar words into one category (run, runs and running becomes one word)

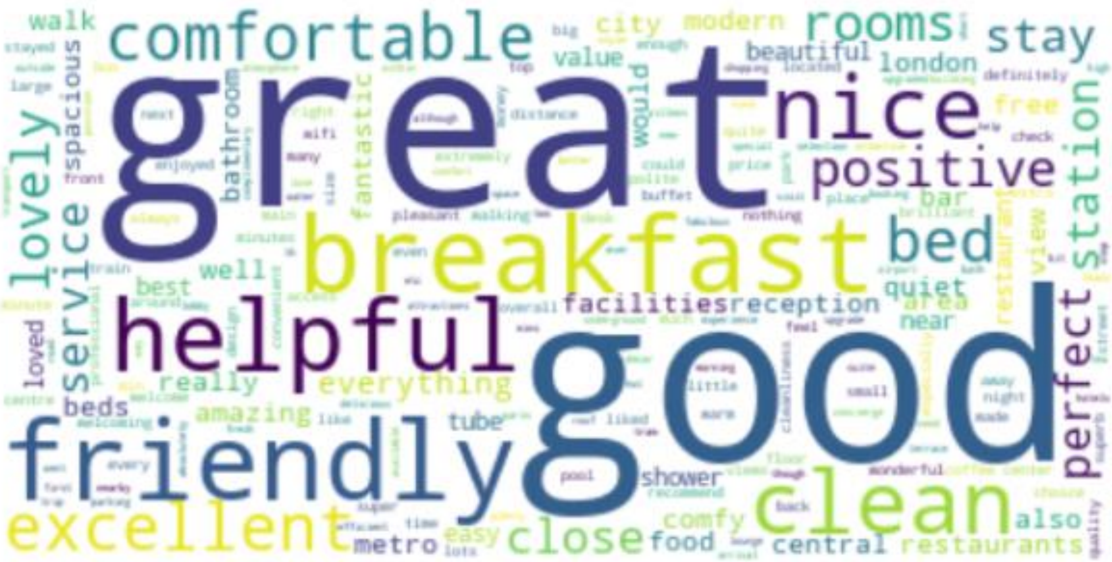
```
#tokenization
df['new_review'] = df['new_review'].apply(lambda x: x.split())
def process(text):
    # Check characters to see if they are in punctuation
    nopunc = set(char for char in list(text) if char not in
string.punctuation)
    # Join the characters to form the string.
    nopunc = " ".join(nopunc)
    # remove any stopwords if present
    return [word for word in nopunc.lower().split() if word.lower() not
in stopwords]

df['new_review'] = df['new_review'].apply(process)
```

In the code snippet below we will once again look for the most frequently used words within negative and positive reviews to get a clear overview of what words are typically used for each sentiment. This will be done by creating “word clouds” for the reviews that are classified negative and the reviews that are classified positive.

```
# function that creates word clouds
def generate_wordcloud(words,colour):
    wordcloud = WordCloud(
    background_color= colour,
    max_words=200,
    stopwords=stopwords
    ).generate_from_frequencies(words)
    plt.figure(figsize=(10,9))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis("off")
    plt.show()
# creating a word cloud based on the negative reviews
neg_words = []
for line in df['new_review'][df['sentiment']==0]:
    neg_words.extend(line)
generate_wordcloud(Counter(neg_words),'black')

# creating a word cloud based on the positive reviews
pos_words = []
for line in df['new_review'][df['sentiment']==1]:
    pos_words.extend(line)
generate_wordcloud(Counter(pos_words),'white')
```

Result:


negative
breakfast
nothing
small
little
bed
bathroom
rooms
could
would
good
bit
like
shower
expensive
really
service
night
stay
water
noisy
reception
bar
floor
also
poor
wifi
time
booking
check
better
coffee
booked
hot
even
need
made
tea
two
door
air
quiet
food
cold
price
area
nice
old
asked
back
double
told
first
outside
facilities
use
work
bad
money
noise
open
dirty
great


great
good
breakfast
friendly
helpful
comfortable
clean
nice
positive
excellent
rooms
stay
lovely
service
station
bed
perfect
close
spacious
bathroom
fantastic
would
facilities
reception
quiet
near
area
staff
walk
city
modern
value
beautiful
london
free
bar
best
well
really
everything
amazing
beds
tube
shower
comfy
food
central
restaurants
metro
easy
also
view
restaurant
loved

For the sentiment analysis the data needs to be trained with a percental amount of our total data. in `test_size` the value is equal to `0.2` meaning that 80% of the reviews will be trained to recognize if a review is negative or positive. In the end the remaining 20%(`0.2`) will be tested to see if the algorithm can indeed predict if a review is negative or positive. There will be an accuracy rate given after every sentimental analysis to see how good the prediction level is/

```
# classification time 
x_train, x_test, y_train, y_test =  train_test_split(df['new_review'],
      df['sentiment'], test_size = 0.2, random_state = 42)
```

Before the data is thrown into the algorithms, the reviews need to be readable for the programs. by applying vectorization which transforms the individual words into a series of 0s and 1s the program can start with the classification procces

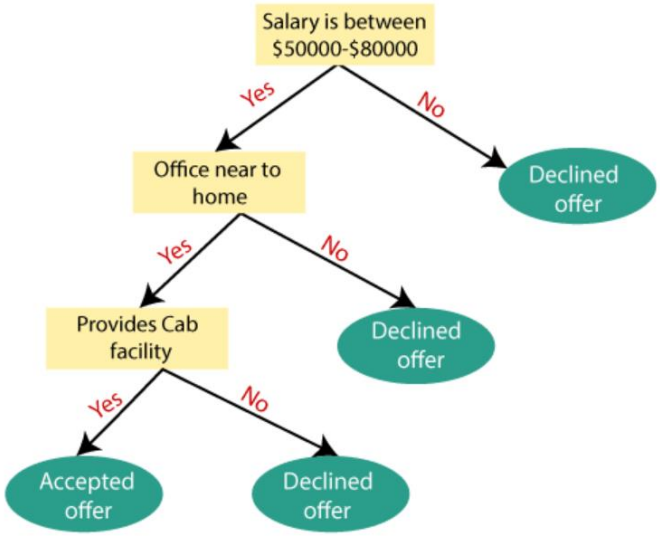
```
count_vect = CountVectorizer(stop_words='english')
transformer = TfidfTransformer(norm='l2',sublinear_tf=True)

x_train_counts = count_vect.fit_transform(x_train)
x_train_tfidf = transformer.fit_transform(x_train_counts)

x_test_counts = count_vect.transform(x_test)
x_test_tfidf = transformer.transform(x_test_counts)
```

	aaargh	ab	abdelsalam	abilities	ability	abit	able	abode	abroad	abrupt	...	zgomot	zl
0	0	0	0	0	0	0	0	0	0	0	...	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
7995	0	0	0	0	0	0	0	0	0	0	...	0	
7996	0	0	0	0	0	0	0	0	0	0	...	0	
7997	0	0	0	0	0	0	0	0	0	0	...	0	
7998	0	0	0	0	0	0	0	0	0	0	...	0	
7999	0	0	0	0	0	0	0	0	0	0	...	0	

8000 rows × 6718 columns

Random forest

A single decision tree is used to categories an object. It this this by going over a checklist with the given object to predict where it belongs. In the decision tree example below it shows if a loan can be given depending if the client meets a certain amount of requirmetns



A “random forest” creates many decision trees and then ask each tree to predict the class value. We could take majority vote and use that answer as our overall prediction. Random forest work on this principle.

n_estimators : This dictates how many decision trees should be built. A higher value will take longer time to run which will lead to higher accuracy.

```
# random forest classification
model = RandomForestClassifier(n_estimators=200)
model.fit(x_train_tfidf, y_train)

predictions = model.predict(x_test_tfidf)

print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
print(accuracy_score(y_test, predictions))
```

In the result below we can see that the total accuracy is 0.8715 which means that that with random forest classification it can accurately predict the sentiment of the review 0.8715% of the time.

	precision	recall	f1-score	support
0	0.87	0.88	0.87	1009
1	0.88	0.86	0.87	991
accuracy			0.87	2000
macro avg	0.87	0.87	0.87	2000
weighted avg	0.87	0.87	0.87	2000

0.8715

Logistic regression

Logistic regression is a statistical method for predicting binary classes. The outcome or target variable is dichotomous in nature. Dichotomous means there are only two possible classes. For example, it can be used for cancer detection problems. It computes the probability of an event occurrence. It is a special case of linear regression where the target variable is categorical in nature. It uses a log of odds as the dependent variable. Logistic Regression predicts the probability of occurrence of a binary event utilizing a logit function.

```
# Logistic regression classification
logmodel = LogisticRegression(random_state=400 )
logmodel.fit(x_train_tfidf,y_train)

log_predictions = logmodel.predict(x_test_tfidf)

print(confusion_matrix(y_test,log_predictions))
print(classification_report(y_test,log_predictions))
print(accuracy_score(y_test, log_predictions))
```

In the result below we can see that the total accuracy is 0.8715 which means that that with random forest classification it can accurately predict the sentiment of the review 0.8715% of the time.

	precision	recall	f1-score	support
0	0.87	0.92	0.89	1009
1	0.92	0.85	0.88	991
accuracy			0.89	2000
macro avg	0.89	0.89	0.89	2000
weighted avg	0.89	0.89	0.89	2000

0.889

Gradient boosting regression

Boosting works on the principle of improving mistakes of the previous learner through the next learner. In boosting, weak learners (ex: decision trees with only the stump) are used which perform only slightly better than a random chance. Boosting focuses on sequentially adding up these weak learners and filtering out the observations that a learner gets correct at every step. Basically, the stress is on developing new weak learners to handle the remaining difficult observations at each step.

```
# GradientBoosting regression classification
alg= GradientBoostingRegressor(n_estimators= 550, learning_rate= 0.1,
max_depth= 3)
alg.fit(x_train_tfidf,y_train)

alg_predictions = logmodel.predict(x_test_tfidf)

print(confusion_matrix(y_test,alg_predictions))
print(classification_report(y_test,alg_predictions))
print(accuracy_score(y_test, alg_predictions))
```

	precision	recall	f1-score	support
0	0.87	0.92	0.89	1009
1	0.92	0.85	0.88	991
accuracy			0.89	2000
macro avg	0.89	0.89	0.89	2000
weighted avg	0.89	0.89	0.89	2000

0.889

conclusion

When used with the scraped data it appears that Random Forest is beaten in terms of accuracy by both logistic regression and gradient boosting.

Reference list

https://www.analyticsvidhya.com/blog/2020/05/decision-tree-vs-random-forest-algorithm/

https://www.mygreatlearning.com/blog/decision-tree-algorithm/

https://www.datasciencecentral.com/profiles/blogs/decision-tree-vs-random-forest-vs-boosted-trees-explained

https://stackabuse.com/python-for-nlp-sentiment-analysis-with-scikit-learn/

https://medium.com/analytics-vidhya/twitter-sentiment-analysis-8ef90df6579c

https://www.zyte.com/learn/what-is-web-scraping/#What-is-web-scraping?

https://www.selenium.dev/documentation/webdriver/

https://datahub.io/docs/data-packages/csv

https://pandas.pydata.org/pandas-docs/stable/user_guide/dsintro.html

http://www.cs.cornell.edu/home/llee/papers/sentiment.home.html

https://learn.g2.com/logistic-regression

https://www.datacamp.com/community/tutorials/understanding-logistic-regression-python

https://www.machinelearningplus.com/machine-learning/gradient-boosting/