DISNEYLAND SENTIMENT ANALYSIS

OBJECTIVE

The aim of this machine learning project is to classify positive and negative sentiments

AGENDA

01

Data description & Data Visualization

02

Operations on text

03

Logistic Regression

04

Word2Vec

05

Word Embedding

06

RNN

Dataset

01

The dataset consists in 42656 observations. Each observation is described by 5 features, and a class (Rating) which defines the mark given by the clients.

	Review_ID	Rating	Year_Month	Reviewer_Location	Review_Text	Branch
0	670772142	4	2019-4	Australia	If you've ever been to Disneyland anywhere you...	Disneyland_HongKong
1	670682799	4	2019-5	Philippines	Its been a while since d last time we visit HK...	Disneyland_HongKong
2	670623270	4	2019-4	United Arab Emirates	Thanks God it wasn t too hot or too humid wh...	Disneyland_HongKong
3	670607911	4	2019-4	Australia	HK Disneyland is a great compact park. Unfortu...	Disneyland_HongKong
4	670607296	4	2019-4	United Kingdom	the location is not in the city, took around 1...	Disneyland_HongKong
5	670591897	3	2019-4	Singapore	Have been to Disney World, Disneyland Anaheim ...	Disneyland_HongKong

Data Cleaning

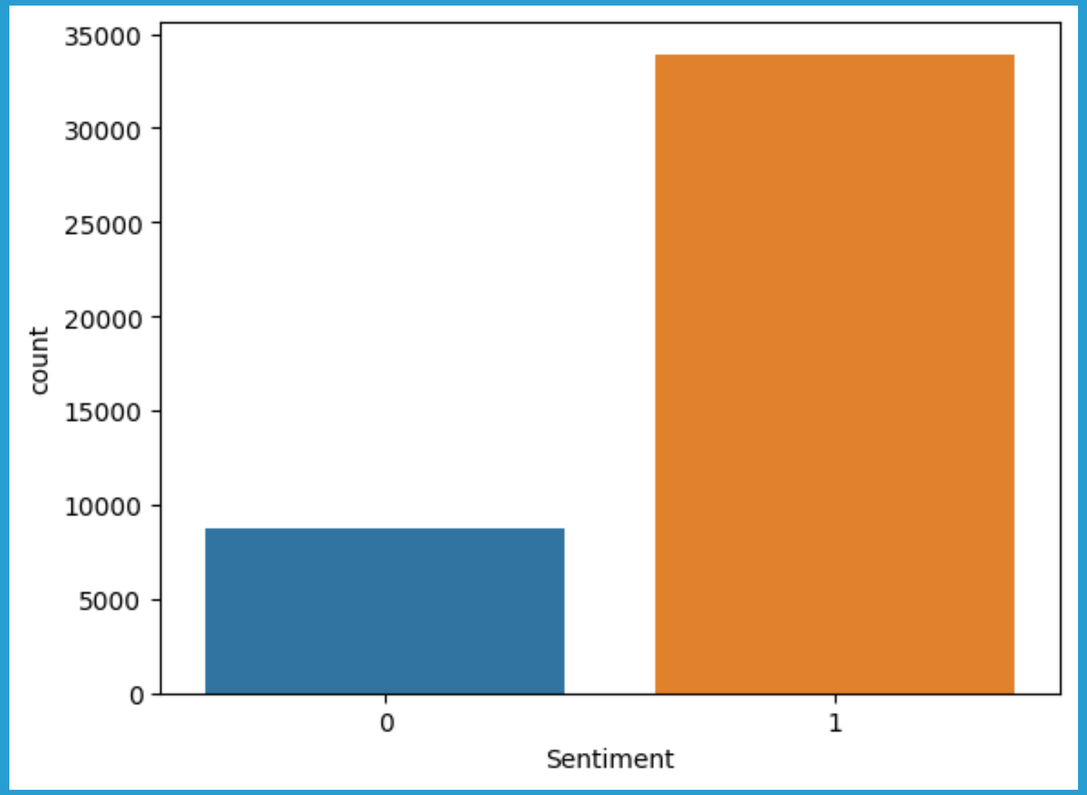
Here I apply a binary classification for the sentiment.
We dentify with 1 the reviews with rating > 3 (this is positive sentiment), and with 0 the reviews with rating < 4 (this is negative sentiment).

```
#insert the sentiment column

Disneyland["Sentiment"] = Disneyland["Rating"].apply(lambda rating : 1 if rating >= 4 else 0)
```

Data Visualization

As we can see, in the dataset, positive sentiments are predominants.



Operations on text

02

Before making the model, I have to do some processes to the text in order to remove unuseful words and to ease the classification process. The procedure is the following:

- Remove stopwords and tokenization
- Remove punctuaction
- Remove htlm strips and noise text
- Remove numbers and special characters
- Stemmatization

Operations on text

02

Stopwords is a function which consists in a set of words (prepositions, articles, verbs, etc). I use this function in order to remove these words from the reviews: that's because these words are extremely frequents in the sentences, and are irrelevant for the analysis.

```
# Remove stop words
def tokenization(text):

    stop_words = set(stopwords.words('english'))
    tokens = text.lower().split()
    text = ' '.join([word for word in tokens if word not in stop_words])
    return text
```

Operations on text


```
string.punctuation
```

✓ 0.0s

```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

```
#Remove punctuation
def remove_punct(text):
    text  = "".join([char for char in text if char not in string.punctuation])
    text = re.sub('[0-9]+', '', text)
    return text
```

Here there are all characters from string.punctuation (a function from the string class).

Here I define the function through which I remove all these characters from the reviews.

Operations on text

02

```
#Removing the html strips
def strip_html(text):
    soup = BeautifulSoup(text)
    return soup.get_text()
#Removing the square brackets
def remove_between_square_brackets(text):
    return re.sub('\[[^]]*\]', '', text)
#Removing the noisy text
def denoise_text(text):
    text = strip_html(text)
    text = remove_between_square_brackets(text)
    return text
```

Here I define the function through which removing html strips and square brackets. To achieve this objective, in the first case I use the function BeatifoulSoup.

Operations on text


```
def clean_text(text):
    # Remove numbers
    text = re.sub(r'\d+', '', text)

    # Remove special characters
    text = re.sub(r'[^\w\s]', '', text)

    return text
```

Here I define the functions to remove numbers and special characters.

Operations on text

02

```
#stemmatization
def stemmer(text):
    ps=nltk.porter.PorterStemmer()
    text= ' '.join([ps.stem(word) for word in text.split()])

    return text
```

Here I create the stemmer function. In this context, I remove all the letters that grammar rules add on words in order to have the same form for each word.

Operations on text

02

I created a new column (“Cleaned_Text”) where I applied all the previous functions in order to have only useful words for the sentiment classification

```
#apply the last function in order to complete the text operations

Disneyland["Cleaned_Text"]=
Disneyland["Cleaned_Text"].apply(stemmer)
```

Frequency Words

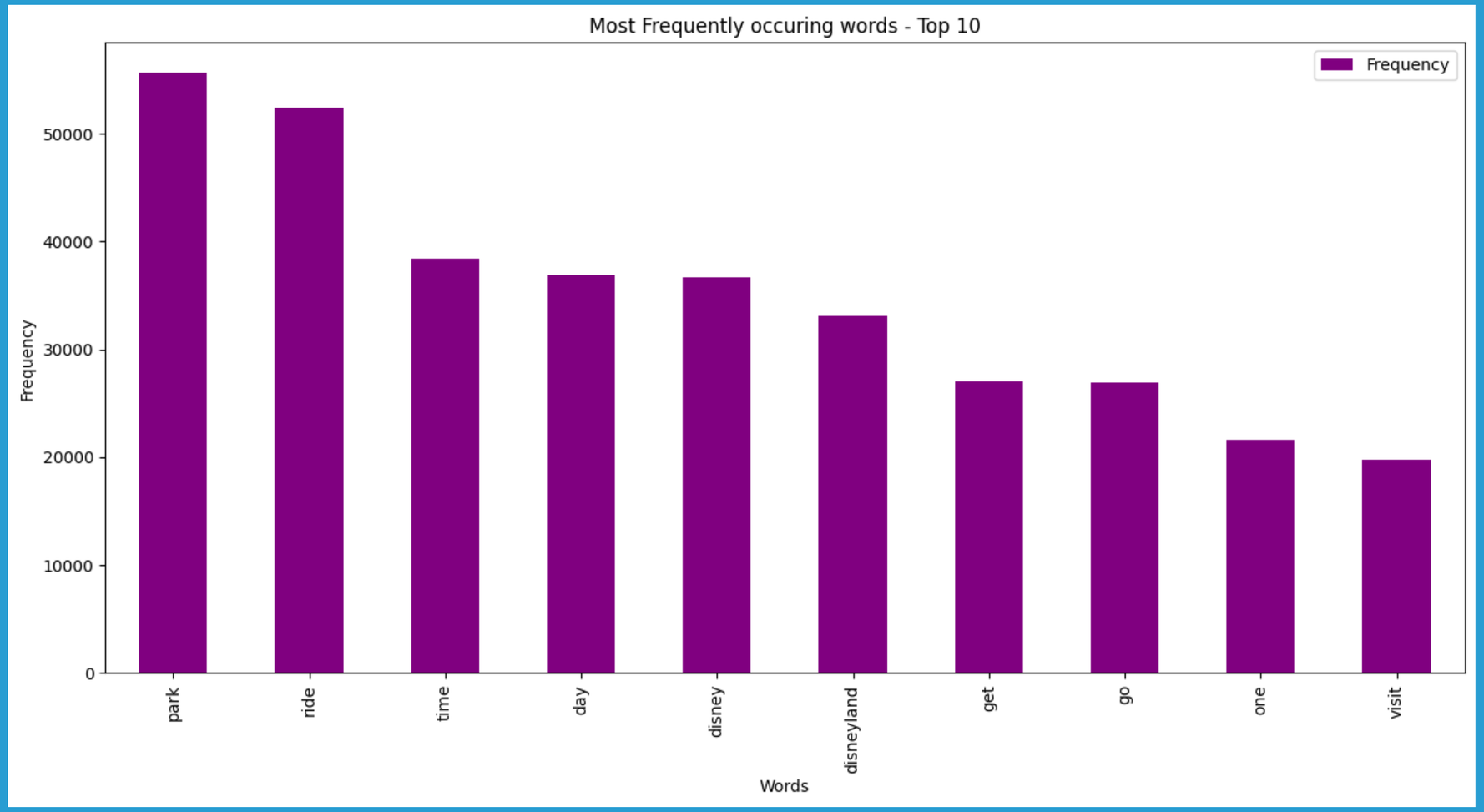
02

```
# Frequency Words
from collections import Counter
words = Counter(' '.join(New_Disneyland['Cleaned_Text'].to_list()).split())
Frequency_words = pd.DataFrame([words]).transpose().reset_index().rename(columns = {'index': 'Words',
0:'Frequency' })
Frequency_words =
Frequency_words.sort_values('Frequency',ascending=False).reset_index(drop='index').reset_index().rename(c
olumns={'index':'Rank'})
Frequency_words['Rank'] = Frequency_words['Rank'].apply(lambda x : x+1)
Frequency_words.head(20)
```

	Rank	Words	Frequency
0	1	park	55678
1	2	ride	52405
2	3	time	38407
3	4	day	36951
4	5	disney	36723
5	6	disneyland	33097
6	7	get	26993
7	8	go	26942
8	9	one	21566
9	10	visit	19763
10	11	wait	16821
11	12	great	16338
12	13	kid	15982
13	14	place	15682
14	15	line	14678
15	16	would	14592
16	17	food	14386
17	18	love	14333
18	19	pass	13863
19	20	show	13326

Frequency Words

02



Frequency Words


```python
from wordcloud import WordCloud
#  WordCloud chart

def show_wordcloud(data, title = None):
    wordcloud = WordCloud(
        background_color = 'black',
        max_words = 350,
        max_font_size = 40,
        scale = 3,
        random_state = 42
    ).generate(str(data))

    fig = plt.figure(1, figsize = (20, 20))
    plt.axis('off')
    if title:
        fig.suptitle(title, fontsize = 20)
        fig.subplots_adjust(top = 2.3)

    plt.imshow(wordcloud)
    plt.show()
show_wordcloud(New_Disneyland["Cleaned_Text"])
```

Logistic Regression

During this second part, due to the different purposes,
I decided to apply three different models:

- Logistic regression
- Word2Vec
- RNN

Logistic Regression

03

I have used the TF-IDF as technique to qunatify words in a set of documents. I decided this technique because through it I could compute the score for each word to highlight its importance into the document and corpus. To adopt the TF-IDF I took the fucntion TfidfVectorizer from sklearn package.

Then, I used the Logistic Regression Model to classify the documents between the two classes (positive or negative sentiment). I adopted a test split of 0.2 . The accuracy of the model is about 86%.

Word2Vec

Between the different models, I choose to use the Word2Vec because this model has an algorithm that get easier understanding the meaning of a word from the context. Here, an hyperparamter plays an important role: the window. This hyperparamter specifies the number of words next to the target word that the algorithm takes into account.

Which option should we consider?

We should consider that machine learning models can’t understand text. We have to convert words into numbers in order to do that. There are 3 options:

1) Convert words into unique numbers basing on the vocabulary

2) One hot encoding

3) Word embedding

Which option should we consider?

With the first option there is the problem that numbers are random, and so this doesn't let me to understand the relationships between words

In the one hot encoding i take all the words, then for each word when I see the word specified we have 1 into the column where there is the selected word and 0 in the rest of the columns. But this way has some issues: it doesn't capture the relationship between the words, and it is computationally inefficient (e.g. if we have a vocabulary with 100.000 words then for each word we have a vector of size 100.000, so the computational cost is very high).

Which option should we consider?

Spoiler: Word embedding

The word embedding consists in a vector of features. Each of them obtains a number between 0 and 1 (higher the number higher the similarity between the feature and the selected word).

Word Embedding

In this section, a person has two options: train your own embedding, or use a pre-trained word embedding (for instance Glove). At the beginning, I took into account the first option, but there are some problems about printing the accuracy of the model (it requires many time, precisely more than one day, and the pc every time turned off). In order overcome this problem, I've decided to use Glove.

Glove

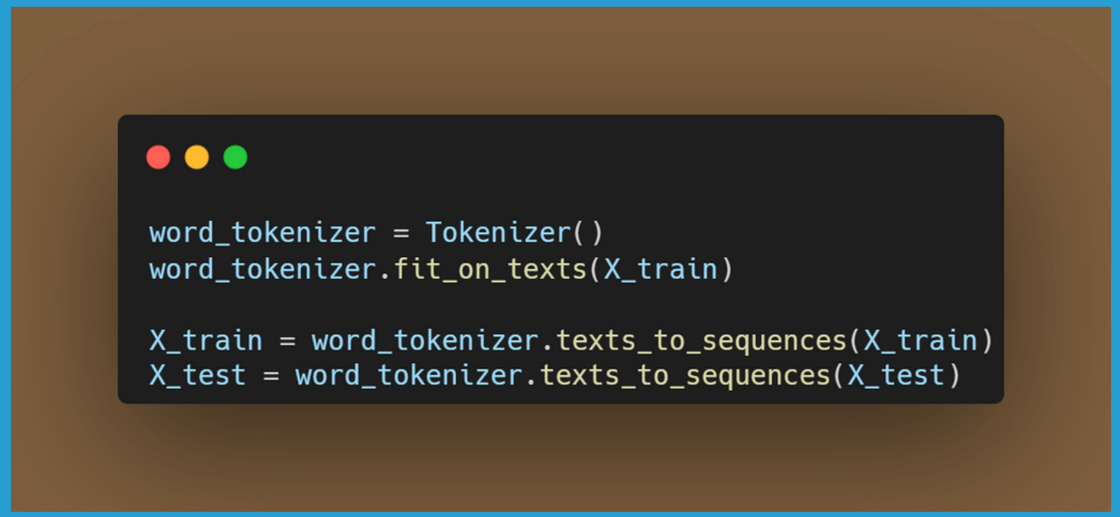
Glove is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word–word co–occurrence statistics from a corpus, and the resulting representations showcase the linear substructures of the word vector space.

Glove

From the different models I choose to use RNNs (precisely, the LSTM) because they have a memory that allows to store and access information from the previous inputs, and that make them the perfect candidate for modelling context and long-term dependencies. Thus, through RNN we can define the sentiment of a sentence and understanding the context by reading it word by word.

Glove

Before loading Glove file, I have to prepare the embedding. Firstly, I split the test into training and set. At the beginning I split in 0.22, but after that I checked with 0.33 and I saw that the latter split improved the accuracy of the model (rnn).



```
word_tokenizer = Tokenizer()
word_tokenizer.fit_on_texts(X_train)

X_train = word_tokenizer.texts_to_sequences(X_train)
X_test = word_tokenizer.texts_to_sequences(X_test)
```

In the next step we use the Tokenizer class in order to create a word–to–index dictionary (where we use each word in the corpus as a key and the corresponding unique index as a value of the key. With the method fit_on_texts I train the tokenizer, while with the method text_to_sequences I convert the sentences into their numeric forms.

Glove


```
vocab_size = len(word_tokenizer.word_index) + 1

maxlen = 100
X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)
```

Then, I create the vocabulary. This is a parameter where to each single word is associated a number (between 0 and the size of the vocabulary) . In this case, the length of vocabulary size is 43003. Then, I put 100 as a maximum length of words for each raw. I saw that usually, this hyperparameter in large datasets could vary from 100 to 400 . To set the length of each raw to the same number I used the pad function, where there are missing values these are replaced with 0.

Glove


```
embeddings_dict = dict()
glove_file = open('glove.6B.100d.txt', encoding="utf8")

for line in glove_file:
    records = line.split()
    word = records[0]
    vector_dimensions = asarray(records[1:], dtype='float32')
    embeddings_dict [word] = vector_dimensions
glove_file.close()
```

After these operations, I open the glove file to load its word embedding and then I create an embedding dictionary. The last operation before making the LSTM model. Here I create an embedding matrix with a number of columns as to the length written before (100).

Then in the last operation before making the LSTM model I create an embedding matrix with a number of columns as to the length written before (100).

LSTM


```
# Neural Network architecture

lstm_model = Sequential()
embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix], input_length=maxlen ,
trainable=False)

lstm_model.add(embedding_layer)
lstm_model.add(LSTM(128))

lstm_model.add(Dense(1, activation='sigmoid'))

lstm_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
```

Now I create the LSTM modell. I put sigmoid as activation function because I wanted to predict the probability. Due to the fact that I have two classes (positive and negative sentiment), I used the binary crossentropy as a loss function.

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_5 (Embedding)     (None, 100, 100)          4300300

 lstm_4 (LSTM)               (None, 128)               117248

 dense_5 (Dense)             (None, 1)                 129

=================================================================
Total params: 4417677 (16.85 MB)
Trainable params: 117377 (458.50 KB)
Non-trainable params: 4300300 (16.40 MB)
_________________________________________________________________
None
```

RNN

06

The last model used in this project is RNN. Why should I choose RNN for sentiment analysis?

There are many reasons for that decision. Firstly is it a neural network that can process sequential data (such as text). In addition they have a memory that allows them to store and access information from previous inputs, which makes them suitable for modeling context (in fact, RNN for sentiment analysis is able to capture the meaning and sentiment of a text by reading it word by word and updating their hidden state accordingly.

Model accuracy

```
Test Score: 0.350551038980484
Test Accuracy: 0.8519570827484131
```

LSTM prediction

After that I load the lstm model created previously, I select the first seven raws from the dataset and test the model on them. I repeat the same process done before with the reviews of the original dataset: I create the column of sentiment from ratings, I clean the text with the functions used before, then I use the tokenizer and finally fix the sentences to a predetermined length (100). Then, I predict the sentiment for these sentences. The final result is the following:

	Review_ID	Rating	Year_Month	Reviewer_Location	Review_Text	Branch	Sentiment	Prediction_value	Sentiment_predicted
0	670772142	4	2019-4	Australia	If you've ever been to Disneyland anywhere you...	Disneyland_HongKong	1	0.875145	1
1	670682799	4	2019-5	Philippines	Its been a while since d last time we visit HK...	Disneyland_HongKong	1	0.736405	1
2	670623270	4	2019-4	United Arab Emirates	Thanks God it wasn t too hot or too humid wh...	Disneyland_HongKong	1	0.942264	1
3	670607911	4	2019-4	Australia	HK Disneyland is a great compact park. Unfortu...	Disneyland_HongKong	1	0.898341	1
4	670607296	4	2019-4	United Kingdom	the location is not in the city, took around 1...	Disneyland_HongKong	1	0.717053	1
5	670591897	3	2019-4	Singapore	Have been to Disney World, Disneyland Anaheim ...	Disneyland_HongKong	0	0.222612	0
6	670585330	5	2019-4	India	Great place! Your day will go by and you won't...	Disneyland_HongKong	1	0.977854	1

Bibliography

- where i found glove link for downloading: https://nlp.stanford.edu/projects/glove/
- book: Texts analystics with Python https://kfsyscc.github.io/attachments/IT/Text_Analytics_with_Python.pdf
- book: Deep Learning for Natural Language Processing https://oku.ozturkibrahim.com/docs_python/Deep_Learning_for_Natural_Language_Processing.pdf