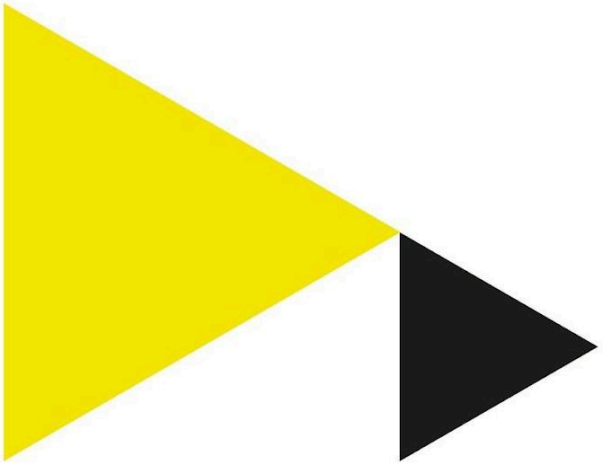
Big Data Scientist and Engineer Assignment II

Deep Learning, Parallel Computing and Visualization techniques applied to Hotel Reviews

Big Data / HBO-ICT / FDMCI

2021 – 2022



Big Data Scientist and Engineer Assignment II

Deep Learning, Parallel Computing and Visualization techniques applied to Hotel Reviews

Author
Hugo Villanueva, #500808745

department
Big Data / HBO-ICT / FDMCI

Date
10-Jan-21

Project type
Assignment

Version
1.0

Abstract

In the previous report, several Machine Learning algorithms were explored to see how effective they were when performing Text-Mining Sentiment Analysis. The same dataset will be used in this report, as well as the same goal, but now using a more sophisticated tool: Deep Learning.

Deep Learning is actually a subset of Machine Learning, but the first is able to process large unstructured data, through the use of its neural networks, and their resemblance to the human's brain of processing new information is more similar than the previous state of Machine Learning.

Aditionally, in order to perform a proper data analysis, it is strongly recommended to design insightful visualisations about the given datasets. In the previous report, some data exploration was made in order to ensure an adequate data cleaning, but now, a visual dashboard will be made in a more user-oriented way, because the data exploration is not necessarily reserved to data analysts or engineers.

Finally, this report also covers some optimisation on the previous Machine Learning techniques by the addition of the Dask library to the scripts, which enables parallel computing, showing how it is possible to enhance the Machine Learning process even more than we explored the previous time.

This new report can show that a dataset is never explored enough, since new techniques keep being developed and the world is filled with data to draw conclusions with.

Table of contents

Abstract 3

Table of contents 4

Introduction 5

1. Data loading 6
- 1.1 Data Sources 6
- 1.2 Data exploration 7
  - 1.2.1 Map-reduce structure 8
  - 1.2.2 Fetching the reviews 9

2. Data visualization 10
- 2.1 Visualization concept 10
- 2.2 Tab 1: Map 10
  - 2.2.1 Visualization mantra 11
  - 2.2.2 Details on Demand 12
  - 2.2.3 App layout 13
  - 2.2.4 App callback 14
- 2.3 Tab 2: Reviews wordclouds 17
  - 2.3.1 Visualization Mantra 17
  - 2.3.2 App Layout 18
  - 2.3.3 App callback 18
- 2.4 Execution 20

3. Parallel Computing for Machine Learning with DASK 21
- 3.1 Theory 21
- 3.2 Dask on Logistic Regression 21
- 3.3 Dask on Hyperparameter Tuning 21

4. Neural Networks 23
- 4.1 Data loading 24
- 4.2 Data preparation 24
- 4.3 Neural Network 1: Convolutional NN 25
  - 4.3.1 Theory 25
  - 4.3.2 Model creation 25
  - 4.3.3 Model evaluation 28
- 4.4 Regular Neural Network 29
  - 4.4.1 Model building 29
  - 4.4.2 Model evaluating 29
- 4.5 Conclusions 31

Bibliography 32

Introduction

The goal of this project is to work further on the Hotel reviews dataset, also researched in the previous paper. However, this time, new advanced techniques will be implemented.

As usual in the Machine Learning life cycle, the first step will be the loading of the data, done in a NOSQL database, MongoDB, that will show inner workings very different from SQL, sometimes more inintuitive, but with some features that add up to the standard SQL model.

Once the data is lodaded in the dataset, it will be collected and transformed to create a visual dashboard with multiple tabs, that allows the user to visualize the data in an interactive way, with filtering, zoom and details by demand of the user.

This visualization can and will potentially help when performing the cleaning necessary for the two differents paths taken in this paper for machine learning. Deep learning with Keras and Machine Learning with Dask.

In the Machine Learning with Dask chapter, the same well-known algorithms from the previous paper will be used, but this time, Dask enables the memory to scale out and perform parallel computing, yielding more optimal results with the same accuracy as the previous version of Machine Learning.

In the Deep Learning chapter, we will take a look at the uses of Neural Networks, a system more complex than traditional Machine Learning, but through the understanding of its concepts, it will be possible to build two neural networks with some components manually added, and see how they perform.

To conclude, these algorithms will be compared based on efficiency and accuracy, and with this information decide how to approach similar problems in the future.

1. Data loading

Before being able to work with the data, it is key to load it into the workspace to ensure that it is comfortable to work with. This time, a NOSQL database will be used, MongoDB. This database model comes with sometimes a harder coding process than vanilla SQL, however, it comes with its advantages that we will see as we advance on the chapter.

1.1 Data Sources

In this project, public data from Hotel reviews websites will be used. For the dashboard, this dataset from Kaggle (https://www.kaggle.com/jiashenliu/515k-hotel-reviews-data-in-europe) contains over 515,000 hotel reviews from different hotels in Europe in english. Each row contains the following fields:

```
df.dtypes ···
Hotel_Address                                object
Additional_Number_of_Scoring                  int64
Review_Date                                  object
Average_Score                               float64
Hotel_Name                                   object
Reviewer_Nationality                         object
Negative_Review                              object
Review_Total_Negative_Word_Counts             int64
Total_Number_of_Reviews                       int64
Positive_Review                              object
Review_Total_Positive_Word_Counts             int64
Total_Number_of_Reviews_Reviewer_Has_Given    int64
Reviewer_Score                              float64
Tags                                         object
days_since_review                            object
lat                                         float64
lng                                         float64
```

Using the libraries PyMongo and Pandas, the dataset will be loaded into MongoDB from the .csv file:

```
# Connection
df=pd.read_csv('Hotel_Reviews.csv')
client = MongoClient("localhost:27017")
db=client.hotel_reviews

# Insertion
# ------------------------------------------------
db.hotel_reviews.insert_many(df.to_dict('records'))
```

After the data is loaded, the next time it is needed, the database can be connected with only these two lines of code:

```
client = MongoClient("localhost:27017")
db=client.hotel_reviews
```

1.2 Data exploration

This data has been already explored in the previous assignment, therefore it makes no sense to plot the same information again. However, the techniques for MongoDB are very different from SQL.

To illustrate, we will plot a graph showing the different nationalities of the reviewers in the dataset. This could be accomplished in SQL with the following code:

```
1    SELECT Reviewer_Nationality, COUNT(*) as count
2    FROM hotel_reviews
3    GROUP BY Reviewer_Nationality
4    ORDER BY count DESC
```

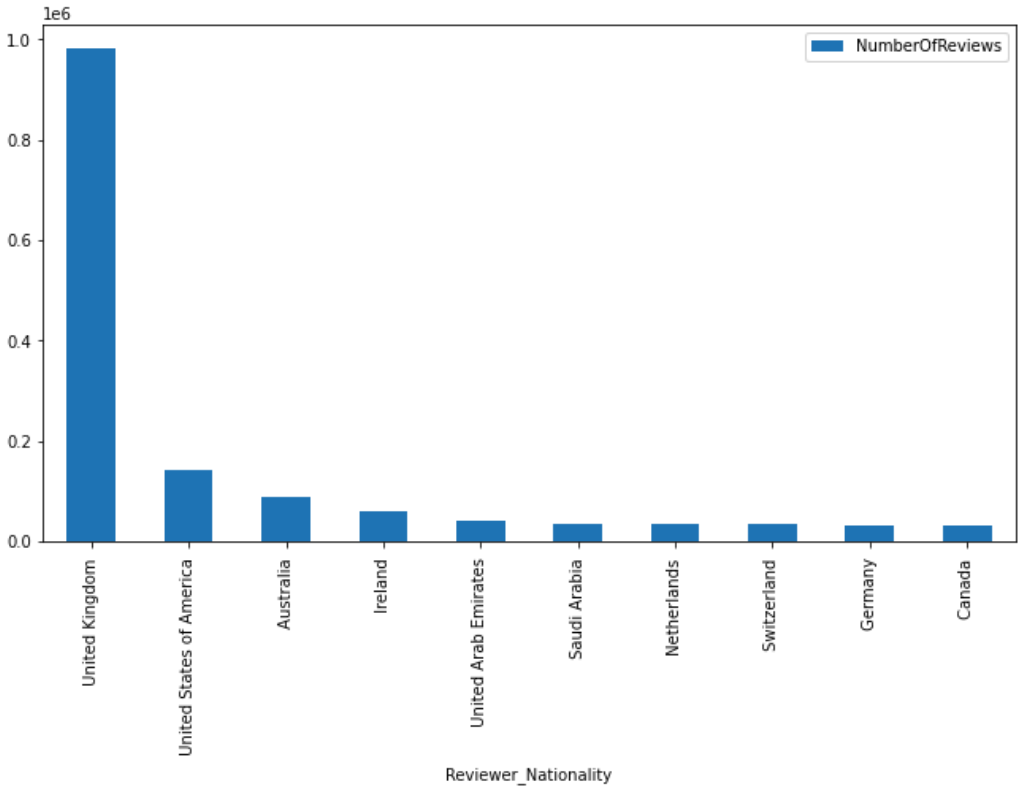
On MongoDB this works quite differently. It is necessary to use the commands $group to group by Reviewer_Nationality, $project to project the number of reviews over them, and then $sort to apply the desired ordering. All this is passed into a pipeline that bears the final result.

```
group={ "$group" : {
                "_id" : {
                    "Reviewer_Nationality" : "$Reviewer_Nationality"
                },
                "Number" : {
                    "$sum" : int(1)
                }
            }
        }
project={"$project" : {
                        "Reviewer_Nationality" : "$_id.Reviewer_Nationality",
                        "NumberOfReviews" : "$Number",
                        "_id" : int(0)
            }
        }

sort={"$sort":{ "NumberOfReviews":-1}}

pipeline=[group,project,sort]
result=db.hotel_reviews.aggregate(pipeline)
```

By converting the result into a Pandas dataframe, we can easily plot the results:

	Reviewer_Nationality	NumberOfReviews
0	United Kingdom	980984
1	United States of America	141748
2	Australia	86744
3	Ireland	59308
4	United Arab Emirates	40940
...	...	...

```
source=list(result)
resultDf=pd.DataFrame(source)
plot = resultDf[:10].plot.bar
(x="Reviewer_Nationality",y="NumberOfReviews",
figsize=(11, 6))
```



1.2.1 Map-reduce structure

This result can also be achieved using the Map-Reduce structure provided by MongoDB, which allows to condense large volumes of data into useful aggregated results, thus saving computing power. This is one of the main advantages of NOSQL.

```
map = Code("function () {emit(this.Reviewer_Nationality, parseInt(1)); }" )
reduce=Code("function (key, values) { return Array.sum(values);}")

resultCol=db.hotel_reviews.map_reduce(map,reduce,"myresult")
resultCol.find_one()
result=resultCol.find({})
```

This code yields the same result as the previous one, though this is more efficient. Since this is a simple aggregate, the difference is less notable, but the pipeline method takes 30.6s and the map-reduce takes 26.8s. This difference is expected to grow as the datasets and/or the queries get bigger.

1.2.2 Fetching the reviews

We intend to use the reviews from the dataset for the creation of the Neural Networks from future chapters, so it is possible to fetch them ussing an aggregate query, which also features a regular expression filter to skip the empty reviews.

```
filter= { "Positive_Review"  :{"$regex":"^\s*"}, "Negative_Review"
 :{"$regex":"^\s*"}}
projection= {
            "Positive_Review" : "$Positive_Review",
            "Negative_Review" : "$Negative_Review",
             "_id" : int(0)
            }
result=db.hotel_reviews.find(filter, projection)
```

```
source=list(result)
sentimentDf=pd.DataFrame(source)
```

This dataframe has two columns. One with positive reviews, and the other with negative reviews. We can transform this into a dataframe whose columns are review, sentiment with the following code:

	Positive_Review	Negative_Review
0	Only the park outside of the hotel was beauti...	I am so angry that i made this post available...
1	No real complaints the hotel was great great ...	No Negative
2	Location was good and staff were ok It is cut...	Rooms are nice but for elderly a bit difficul...
3	Great location in nice surroundings the bar a...	My room was dirty and I was afraid to walk ba...
4	Amazing location and building Romantic setting	You When I booked with your company on line y...
...	...	...

review	label
Only the park outside of the hotel was beauti...	POSITIVE
No real complaints the hotel was great great ...	POSITIVE
Location was good and staff were ok It is cut...	POSITIVE
Great location in nice surroundings the bar a...	POSITIVE
Amazing location and building Romantic setting	POSITIVE
...	...

```
# Transform into review/sentiment format
df_pos=sentimentDf[(sentimentDf["Positive_Review"]!="No
Positive")&(sentimentDf["Positive_Review"]!="Nothing")]
df_neg=sentimentDf[(sentimentDf["Negative_Review"]!="No
Negative")&(sentimentDf["Negative_Review"]!="Nothing")]

df_pos = df_pos.assign(label='POSITIVE')
df_neg = df_neg.assign(label='NEGATIVE')

df_pos=df_pos.rename({"Positive_Review": "review"}, axis="columns")
df_neg=df_neg.rename({"Negative_Review": "review"}, axis="columns")

sentimentDf = pd.concat([df_pos,df_neg])
sentimentDf = sentimentDf[["review","label"]]
sentimentDf = sentimentDf.drop_duplicates()
```

We can load this new dataframe into a new database, so it can be ready for future use right away.

```
db.Sentiment_Reviews.insert_many(sentimentDf.to_dict('records'))
```

Now we have everything ready to start working with the data for the Machine Learning and Deep Learning algorithms.

2. Data visualization

Data visualization is a key step on the processing of every dataset. In this example, a client-oriented map dashboard will be developed using Plotly and Dash libraries. Dash works on a layout-callback basis, so there will be a separate chapter for each coding section.

2.1 Visualization concept

As seen before, the dataset contains hundreds of different hotels and each one with hundreds of different reviews. The size of the dataset is quite large to make a map visualization than runs smoothly on a home computer. Aditionally, the dataset features hotels for several capitals in Europe, so a general map would be empty for the most part, except in the capitals, where it would be crowded.

For this reasons, a subset of the dataframe will be used, containing only Hotels from Amsterdam, thus making the map be zoomed in to only select this region. This decision also allows an immediate visualization of geographically distributed data points. The querying of only Amsterdam hotels has been done with the following code:

```
# All reviews from Amsterdam
amsdf=df[df['Hotel_Address'].str.contains('Amsterdam Netherlands') == True]
# Get one review per hotel, hence getting unique hotels
hotelsdf=amsdf.drop_duplicates(subset=['Hotel_Address'])
```

Since all adresses contain the city and the country at the end of it, it is possible to filter a certain city by looking for a substring with this method. Note that we did not use only ‘Amsterdam’ because some hotels in the rest of Europe feature Amsterdam as its hotel name.

Once the data required is ready to show, the goal of the visualization is to show in a map of Amsterdam with its hotels and its average score (provided by the Average_Score in the dataset). It also features a filter so that the user can select a specfic score range.This could be summed up in a user story as: “As a visitor in Amsterdam, I would like to be able to select a Hotel with a certain score in a certain area”

Then, the dashboard contains another tab, where two wordclouds are shown containing either positive or negative reviews. The user can select wordclouds for a specific hotel, and thus get a quick idea of the good and bad things in this particular hotel.

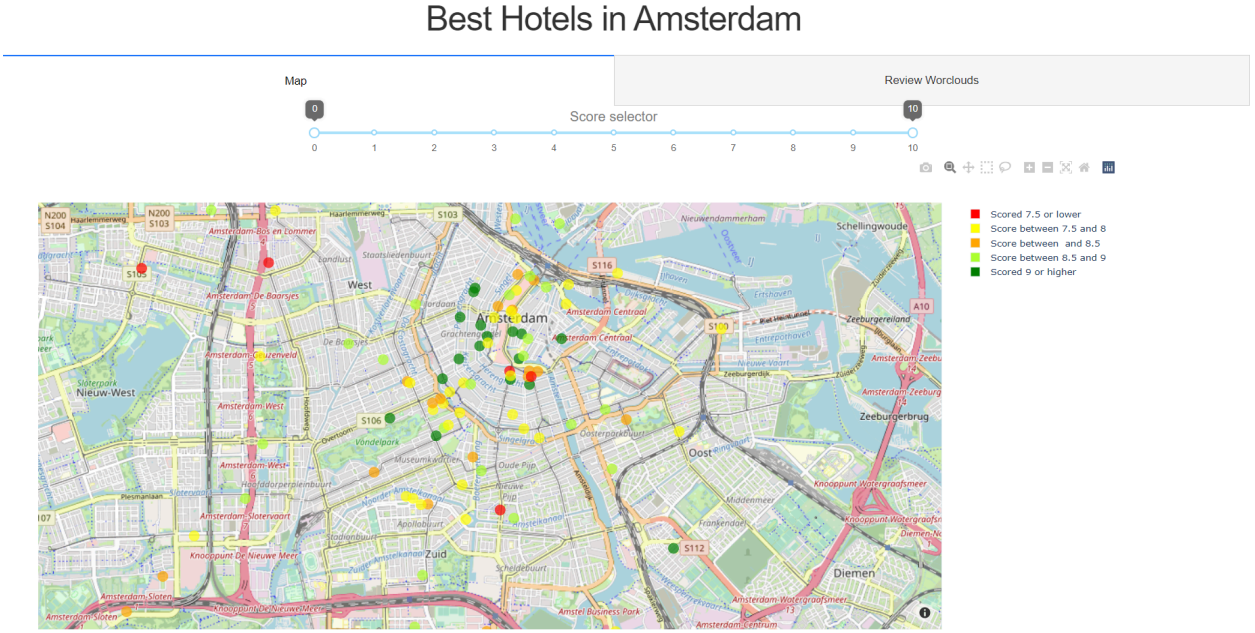
2.2 Tab 1: Map

2.2.1 Visualization mantra

According to Schneiderman’s Mantra of Visualization, every visualization system should follow the following principles, as this visualization does:

Overview first

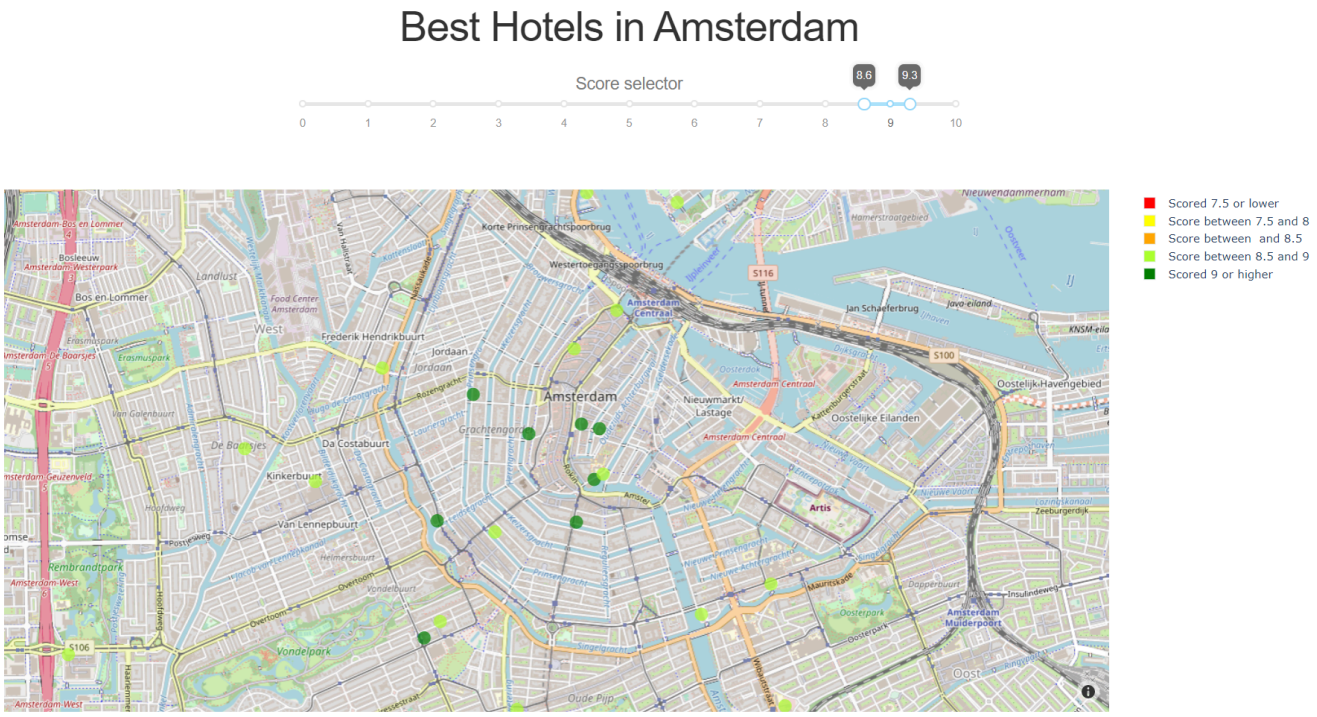
When the application is loaded, the whole map is shown on the screen, showing all the hotels and their color code for their score.



Zoom and Filter

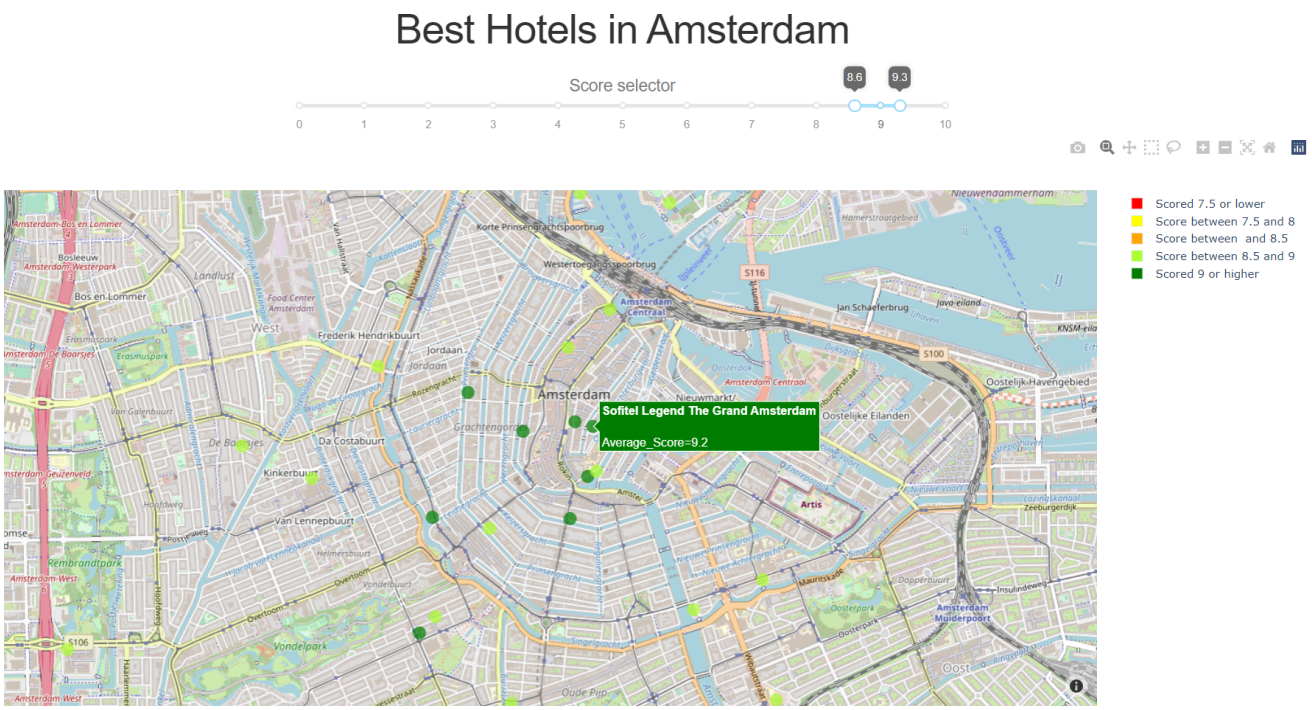
The map uses plotly graphs techonology, which easily allows the user to zoom in and out on the plot, as well as select a certain region to zoom in, and double-click to zoom out.

The app also provides a score filter, using a double score silder from 0 to 10 (which by default selects everything), and it can be customized to select any specific range, including decimals. The map updates immediately as the slider is moved.



2.2.2 Details on Demand

In order to allow the user to explore specific data points without cluttering the screen, the graph features a tooltip option that activates when the user hovers the mouse over a certain dot on the map. It shows the name of the hotel as well as its average score.



2.2.3 App layout

In Dash, the app layout contains the HTML elements, with static elements, usually for style, and empty containers, that have unique IDs so that the callback can fill them with dynamic data. In this dashboard, the title and the slider are static, and the map is dynamic, because it changes according to the selection of the filter.

HTML elements

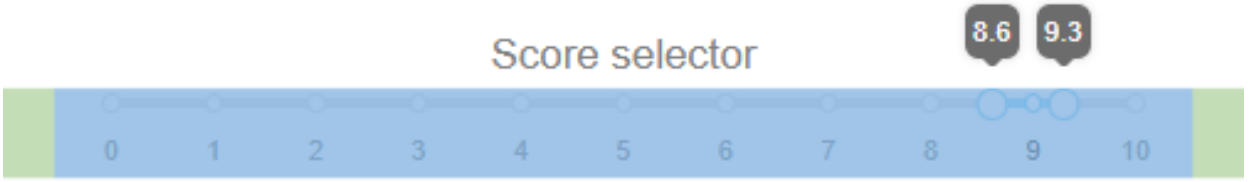
Dash can contain a vast variety of HTML elements. For demonstration purposes, this dashboard only contains a H1 and Divs, but also experiments with CSS style for position and styling throughout the whole layout.

```
    # Title
    html.H1('Best Hotels in Amsterdam',  style={'padding-left':'25%',
'padding-right':'25%', 'text-align':'center'}),
```

The slider

The slider is emebedded in a div container, which is below another div container containing the title of the slider (Score selector) that uses CSS for position and styling.

```
 # Slider
    html.Div('Score selector',
 style={'padding-left':'25%','padding-right':'25%',
'text-align':'center','fontSize': 18,'color': 'gray', }),
```



The slider itself is made with the object dcc.RangeSlider, which is a double input and double output slider (int array). Thanks to the step parameter, it is possible to select decimal values in the slider, allowing the user a more specific filtering. Aditionally, it has the tooltip always active so that the user always knows which is their exact input into the filter.

```
    html.Div([
        dcc.RangeSlider(
        id='score-slider',
        min=0,
        max=10,
        value=[0, 10],

marks={"0":0,"1":1,"2":2,"3":3,"4":4,"5":5,"6":6,"7":7,"8":8,"9":9,"10":10},
        step=0.1,
        tooltip = {'always_visible': True }
        ),
        ],
        style={'width': '50%','padding-left':'25%', 'padding-right':'25%'}
    ),
```

Graph container

In the app layout, the object dcc.Graph must be placed in order to create a container to receive the Dash callback. Since it is a dynamic object, no figure should be placed in this graph, only the id to receive the callback.

```
    # Graph
    dcc.Graph(id='my_hotel_map')
```

2.2.4 App callback

The function of the callback is to plug in the layout the elements desired, through inputs received in the layout and outputs processed in this layer.

In this application, the input is the score slider, and the output is the hotels map.

```
# Connect the Plotly graphs with Dash Components
@app.callback(
    Output("tabs-content-example", "children"),
    Input("tabs-example", "value"),
    Input("score-slider", "value"),
    Input("dropdown", "value"),
)
```

```
def render_content(tab, selected_score=[0, 10], hotel_name="All"):
```

The input is processed with a function that received the selected score as a parameter, and queries the dataset to get the desired data. This function can also render the Reviews tab, chosen by the tab parameter.

```
def render_content(tab, selected_score=[0, 10], hotel_name="All"):

    # Tab 1: Map
    if tab == "tab-1-example":

      filtered_df  =  hotelsdf[(hotelsdf["Average_Score"]  >  selected_score[0])  &
(hotelsdf["Average_Score"] < selected_score[1])]
```

Now, the filtered_df contains all the data required by the plot to draw the graph.

Figure plotting

The figure is plotted using Plotly Express and Mapbox libraries. The fields of the dataframe filtered_df (wich are the same as in the original dataset) are passed as parameters, such as the latitud and longitude used to place the dots on the map, and the score and the name to use as tooltip information. Extra parameters as zoom, width and height are stated to ensure a comfortable initial view of the dashboard.

```
fig = px.scatter_mapbox(data_frame=filtered_df,
                        lat=filtered_df["lat"],
                        lon=filtered_df["lng"],
                        hover_name=filtered_df["Hotel_Name"],
                        zoom=12.2,
                        width=1500,
                        height=700,
                        hover_data=dict(
                                    Hotel_Name=False,
                                    Average_Score=True,
                                    lat=False,
                                    lng=False),
                        )
```

Traces coloring

However, the previous code does not add differentiation to the data points, so in order to customise them, we can use the method .update_traces.

```
    fig.update_traces(marker=dict(
                size=15,
                opacity=0.7,
                color=list(map(SetColor, filtered_df['Average_Score']))))
```

The color of the data points is defined by an external function which defines the boundaries for each color, and this function is then called and mapped into the traces.

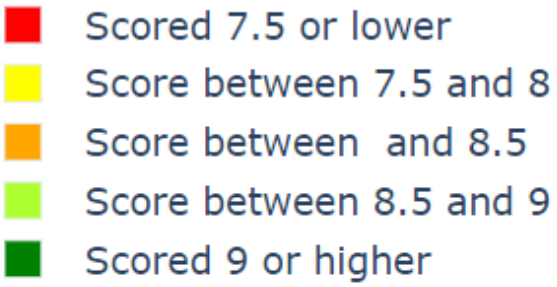
```
# Function for data points coloring
def SetColor(x):
        if(x < 7.5):
            return "red"
        elif(7.5<= x <=8):
            return "orange"
        elif(8< x <=8.5):
            return "yellow"
        elif(8.5< x <=9):
            return "GreenYellow"
        elif(x > 9):
            return "green"
```

Legend

Since the coloring of the traces has been used an external function, the traditional and simple method for making the legends of activating the legend parameter on the plot does not work. So instead, invisible traces have been added to the map using the same color function as before (SetColor) and then these traces can be automatically shown in the legend.

```
# Legend
fig.add_trace(go.Bar(name="Scored 7.5 or lower", marker=dict(
            color=list(map(SetColor, hotelsdf['Average_Score']))[26]),x=[""], y=[""]))
fig.add_trace(go.Bar(name="Score between 7.5 and 8", marker=dict(
            color=list(map(SetColor, hotelsdf['Average_Score']))[7]),x=[""], y=[""]))
fig.add_trace(go.Bar(name="Score between  and 8.5", marker=dict(
            color=list(map(SetColor, hotelsdf['Average_Score']))[0]),x=[""], y=[""]))
fig.add_trace(go.Bar(name="Score between 8.5 and 9", marker=dict(
            color=list(map(SetColor, hotelsdf['Average_Score']))[2]),x=[""], y=[""]))
fig.add_trace(go.Bar(name="Scored 9 or higher", marker=dict(
              color=list(map(SetColor, hotelsdf['Average_Score']))[28]),x=[""], y=[""]))
```

Each one of the new traces contains one hotel from the Amsterdam hotels datarame, corresponding to one of the score ranges indicated in the coloring function. Then they are set as invisible traces (x and y parameters empty) so they do not appear in the graph as extra points. This allows the legend to contain the exact same colors as the traces, without explicitly defining them again.

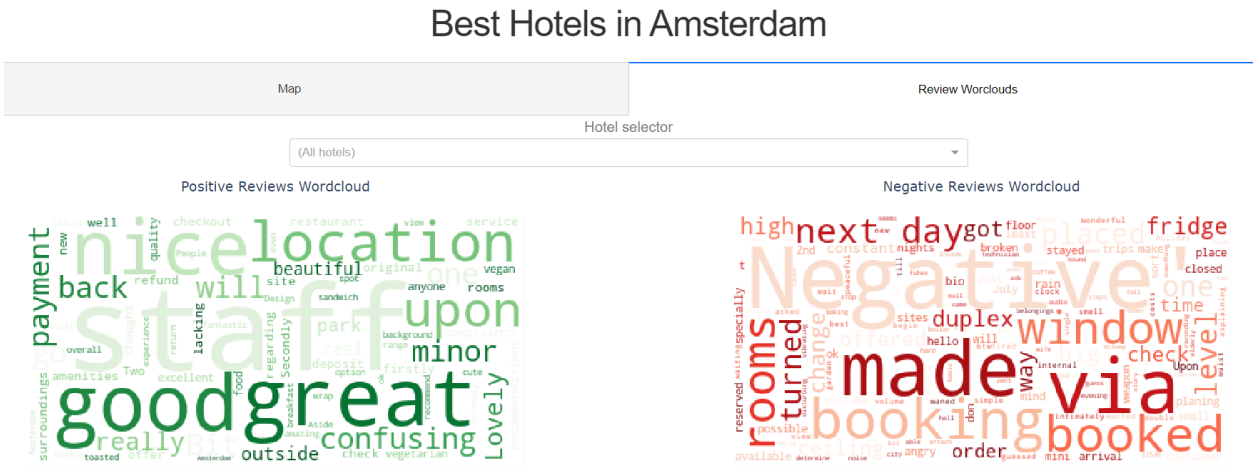


2.3 Tab 2: Reviews wordclouds

2.3.1 Visualization Mantra

Overview

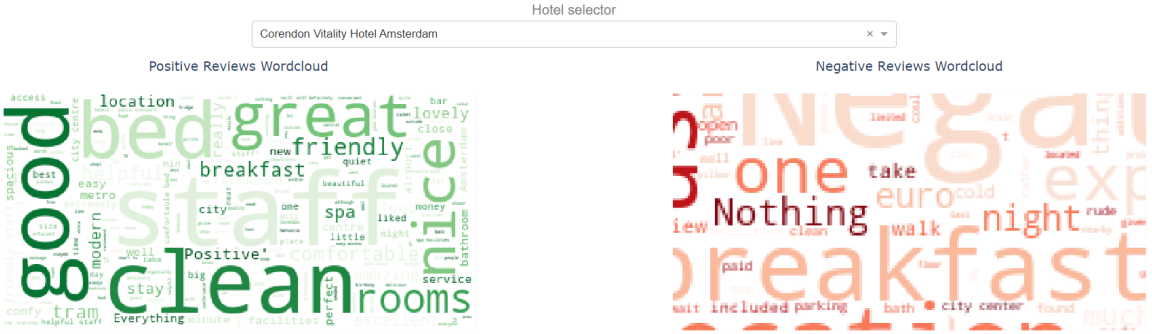
When the tab is opened, the default wordclouds display a summary of all the hotel reviews combined, thus creating an overview wordcloud:



Zoom and Filter

Since both wordclouds are Dash graphs, have the zoom option:

The filter feature is available through the dropdown menu, which allows the user to select a Hotel from the list (the same as in the map) and display wordclouds for the reviews from this hotel:



Details on Demand

Since the goal of a wordcloud is precisely to avoid showing details, and rather a general sense of the reviews, the details-on-demand feature could be considered to be the hotel selection as well.

2.3.2 App Layout

This tab follows the same structure as the previous one, but this time with a Dropdown menu instead of a slider. The Dropdown options are values are the same: The hotel names, which are fetched through a for loop through the dataframe. The placeholder (the text that appears before selecting any option) displays the text “(All hotels)” to inform the user that if nothing is selected, the whole dataset is plotted.

```
                dcc.Tab(
                    label="Review Worclouds",
                    value="tab-2-example",
                    children=[
                        html.Div(
                                [
                                 dcc.Dropdown(
                                     id="dropdown",
                                     options=[
                                         {"label": i, "value": i}
                                         for i in hotelsdf.Hotel_Name.unique()
                                     ],
                                     multi=False,
                                     placeholder="(All hotels)",
                                 ),
```

2.3.3 App callback

The callback for the tab is made if the selected tab is the second (indicated by the variable “tab”)

```
# Tab 2: Worcloud
    elif tab == "tab-2-example":
```

Then, the data for the chosen hotel must be filtered, unless there is no hotel selected , and in that case, the dataframe will contain all reviews from Amsterdam:

```
        selected_reviewsdf = amsdf[amsdf["Hotel_Name"] == hotel_name]

        # Wordcloud with all hotels, if none chosen
        if len(selected_reviewsdf) < 1:
            selected_reviewsdf = amsdf
```

Now that we have the required dataframe for the wordclouds, it is time to start working with them.

Wordcloud plotting

Luckily, the dataset has the positive and negative reviews in different columns, so it is easy to create variables containing the text from each of them.

```
        neg_text = selected_reviewsdf["Negative_Review"].values
        pos_text = selected_reviewsdf["Positive_Review"].values
```

The most common words in a wordcloud are usually the least informative ones. We are taking out the default stopwords as well as the most common (and boring) words in the reviews, as shown below.

```
        stopwords = set(STOPWORDS)
```

```
        stopwords.update(["hotel", "room", "positive", "Negative"])
```

Then, the wordclouds are generated using the WordCloud library. Giving the stopwords as a parameter, the algorithm ignores them in the plotting. It is also possible to change the background color to white, so that it matches the background of the dashboard.

```
        neg_wordcloud = WordCloud(stopwords=stopwords,background_color='white').generate(str(neg_text))
        pos_wordcloud = WordCloud(stopwords=stopwords,background_color='white').generate(str(pos_text))
```

If a specific range of colors is desired for the wordclouds, it is possible to recolor them using the default colormaps provided by WordCloud. This way, it will be visually clear which wordcloud belongs to which sentiment.

```
        neg_wordcloud.recolor(colormap="Reds", random_state=1)
        pos_wordcloud.recolor(colormap="Greens", random_state=1)
```

Now, the wordclouds must be put into a plotly graph. Using this library, it is possible to remove the axes and adding a custom title.

```
        neg_fig = px.imshow(neg_wordcloud)
        neg_fig.update_xaxes(showticklabels=False).update_yaxes(showticklabels=False)
        neg_fig.update_layout(title_text="Negative Reviews Wordcloud", title_x=0.5)

        pos_fig = px.imshow(pos_wordcloud)
        pos_fig.update_xaxes(showticklabels=False).update_yaxes(showticklabels=False)
        pos_fig.update_layout(title_text="Positive Reviews Wordcloud", title_x=0.5)
```

Finally, the figure is embedded into a Dash div, which will be put into the tab renderer. Since the wordclouds are not very wide, it is interesting to have them side by side, also allowing easier comparison. It is possible to do that by making a column-row HTML structure.

```
        tab_content = html.Div(
            [
                html.Div(
                    [
                        html.Div(
                            [
                                dcc.Graph(id="graph-2-tabs", figure=pos_fig),
                            ],
                            className="six columns",
                        ),
                        html.Div(
                            [
                                dcc.Graph(
                                    id="graph-3-tabs",
                                    figure=neg_fig,
                                ),
                            ],
                            className="six columns",
                        ),
                    ],
                    className="row",
                )
            ]
        )

    return tab_content
```

2.4 Execution

After a successful connection between the layout and the callback, the app can be run with the classic if clause:

```
# Run
if __name__ == '__main__':
    app.run_server(debug=False)
```

It runs on a local server (127.0.0.1), using by default the port 8050. The IDE shows a 200 OK code everytime the callback is successful when refreshing the map.

```
127.0.0.1 - - [14/Jan/2022 21:40:41] "POST /_dash-update-
component HTTP/1.1" 200 -
```

3. Parallel Computing for Machine Learning with DASK

Dask is an open source library for parallel programming and distributed computing in Python created by Matthew Rocklin in late 2014, designed to parallel computing and delaying tasks.

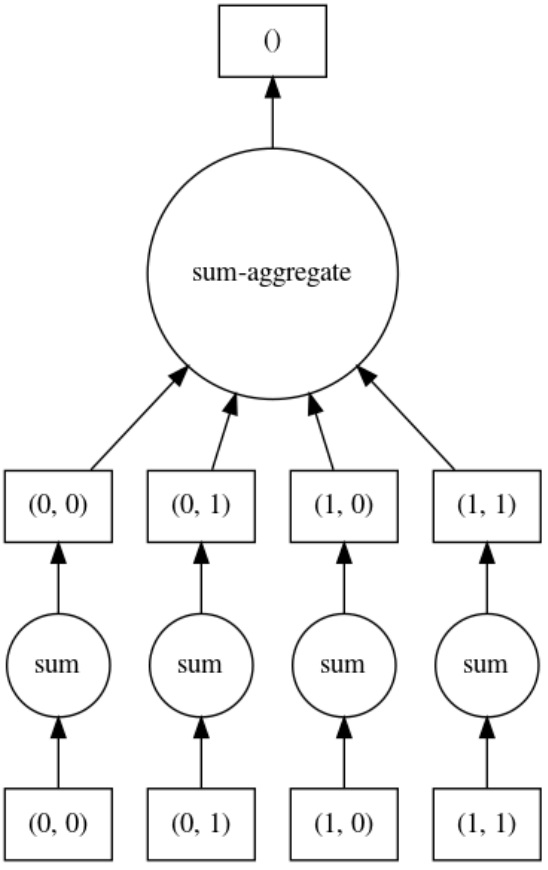
3.1 Theory

Dask is a Python library that allows is users to scale out their programs, which helps reducing the execution time. This can potentially save tremendous amounts of time, specially when it comes to typically slow algorithms like the ones used in Machine Learning.

The main principle of Dask is to create parallel clusters in the computer's Random Access Memory (RAM), so that each one of them can work independently. Then, Dasks breaks down the big tasks into pieces and assigns them into each one of the clusters, and then combine the results.

This principle is called scaling out, and it is the logical evolution from scaling up, wich meant simply upgrading the capacity of the memory itself. With parallel computing, several mediocre memories together can do the same job as a good memory.

This figure represents how Dask has broken down the sum-aggregate into 4 blocks ((0,0),(0,1),(1,0),(1,1)), whose sum is processed independently, and their results are then aggregated in a single instance, giving the final output in a reduced time, compared to doing it in a serial way.



For this experiment, a smaller portion of the dataset will be used to ensure that it can be ran in a single day. Hence the time results will be dissimilar from a normal Dask Machine Learning algorithm.

Dask is not the only option when resorting to parallel computing. There are other famous options like Spark. However, Dask provides visual tools and better Python integration, as well as with other commonly used libraries in Data Science like NumPy (arrays) and Pandas (Dataframes)

3.2 Dask on Logistic Regression

In this chapter, we will run a Logistic Regression Machine Learning on our hotel review dataset, using both traditional sklearn library and Dask built-in Logistic Regression. Both methods ensure the same results, but the execution time will differ. For timing the results, the library provided by timeit will be used.

Sklearn logistic regression:

```
sklearn_log_reg=sklearn.linear_model.LogisticRegression()
t1 = tic()
sklearn_log_reg.fit(X_train, y_train)
timings.append(('sklearn', tic() - t1))
```

Dask logistic regression:

```
dask_log_reg=dask_ml.linear_model.LogisticRegression()
t1 = tic()
dask_log_reg.fit(X_train, y_train)
timings.append(('dask-ml', tic() - t1))
```

If we print the timings list, it is indeed possible to see the difference:

```
[('sklearn', 7.306704599999648), ('dask-ml', 0.9153105999998843)]
```

3.3 Dask on Hyperparameter Tuning

Dask is also possible to use when doing GridSearchCV while looking for the optimal configuration of the parameters.

```
hypertimings=[]

param_grid = {
    'C': [0.001, 1, 10.0, 100.0],
}
grid_search = GridSearchCV(sklearn_log_reg, param_grid, verbose=1, cv=4,
n_jobs=-1)

# Vanilla GridSearch
t1 = tic()
grid_search.fit(X, y)
hypertimings.append(('sklearn', tic() - t1))

# Dask Gridsearch
with joblib.parallel_backend("dask"):
    grid_search.fit(X, y)
hypertimings.append(('dask', tic() - t1))
```

Checking the results, indeed Dask proves to be a better alternative:

```
[('sklearn', 1.876564999999573), ('dask', 0.2639995999998064)]
```

3.4 Conclusions

Execution times of Sklearn VS Dask

	Model Training	Hyperparameter Tuning
Sklearn	7.3s	1.87s
Dask	0.91s	0.26s

Dask is a very powerful tool with a lot of possibilities, and it can perform just as well as traditional Machine Learning algorithms without any drawbacks, apart from the need for getting to learn to use it, but the theory behind the library is not extremely complex and the visual tools help the library to be user-friendly enough to be used by newcomers to Machine Learning.

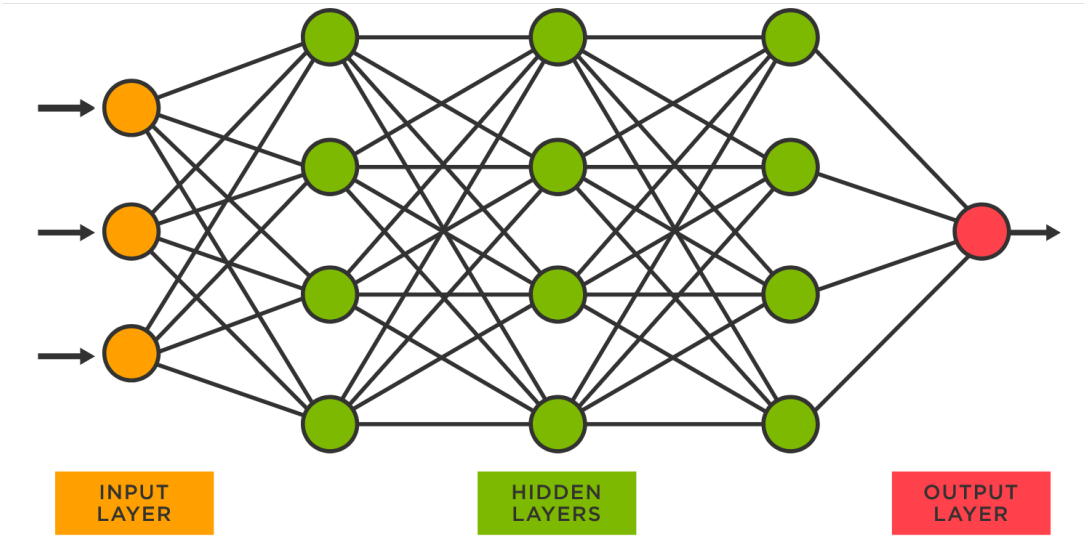
However, the Sklearn traditional algorithms do not have a dramatically bad performance overall, and, depenending on the context, could also be used without parallelizing, since they get the job with the same accuracy, and once the model is trained, it can also be stored so it is not necessary to be executed again, with libraries such as Pickle.

Dask means a big step forward in Machine Learning, and if the amount of computers subscribed to a system can keep increasing easily, it will not take long before we get to see astoundingly big computations done in seconds thanks to parallel computing.

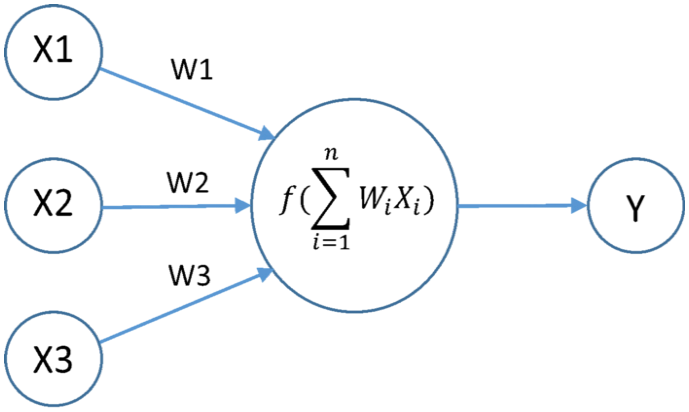
4. Neural Networks

A Neural Network is a computing system inspired by the biological neuron system present in the human mind. It is based in a collection of nodes, also called artificial neurons, which are interconnected with synapses and send signals between themselves.

Each neuron has an entry point and an exit. The first layer of the network is called the Input Layer, that received the raw data and passes it onto the next layer after the first step in the processing.
After going through a link, a weight function is applied to the result, which can either increase or inhibit the activation state of adjacent neurons. This process carries over through a determined amount of layers in the core of the network, called Hidden Layers. Finally, the output layers condenses the results of the last hidden layer into one solution.



The output of the nodes is also limited by the activation function, which, if exists, modifies the output result from the previous layers or caps it before propagating to the next layer. This may be helpful to stop propagating anomalies in certain neurons that could potentially carry on to a faulty model.



There are many possibilities for the activation functions, but the most common ones are the Linear, ReLU, Heaviside and Logistic. The best fitting one for our model will be discused.

Finally, after the whole network has completed its process, the accuracy of the model is calculated by a Loss Function, that uses backpropagation through the network with respect to the weights of the neurons to calculate the inaccuracy of each propagation.

4.1 Data loading

As done before, the data will be fetched from the NOSQL database, using the Sentiment_Reviews database created in the first chapter.

```
client = MongoClient("localhost:27017")
db=client.hotel_reviews
```

```
result=db.Sentiment_Reviews.find({})
source=list(result)
df=pd.DataFrame(source)
df=df[["review","label"]]
```

A subset of 100,000 reviews will be used in this experiment, since using the whole dataset could have major drawbacks on execution time, though it would be recommended to use the whole dataset if the maximum accuracy is desired. For demonstration purposes, 100,000 is a good number to get good enough results.

```
df=df.sample(100000,random_state=1)
```

4.2 Data preparation

Text preprocessing

Since most texts require the same steps in preprocessing, this function provided by the teachers does the job perfectly when removing undesired symbols in the text. It gets the dataset ready with only words.

```
def remove_tags(text):
    return TAG_RE.sub('', text)
def preprocess_text(sen):
    # Removing html tags
    sentence = remove_tags(sen)
    # Remove punctuations and numbers
    sentence = re.sub('[^a-zA-Z]', ' ', sentence)
    # Single character removal
    sentence = re.sub(r"\s+[a-zA-Z]\s+", ' ', sentence)
    # Removing multiple spaces
    sentence = re.sub(r'\s+', ' ', sentence)
    return sentence
X = []
sentences = list(df['review'])
for sen in sentences:
    X.append(preprocess_text(sen))
```

Since the reviews are labeled with strings (“POSITIVE”/”NEGATIVE”), it is necessary to map them into numeric values, which is how Machine Learning processes sentiment in text. 1 will be positive and 0 will be negative.

```
y = df['label']
y = np.array(list(map(lambda x: 1 if x=="POSITIVE" else 0, y)))
```

The train/test split, the usual step in Machine Learning, will be donde with a test size of 20%, since Neural Networks make good use of a bigger data feeding to the algorithm.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=1)
```

Tokenization

As in traditional Machine Learning, the text must be tokenized, creating, by defninition, “an instance of a sequence of characters in some particular document that are grouped together as a useful semantic unit for processing.” (Manning et al., 2008).

Keras provides a built-in tokenizer that can be used right away in the code:

```
X_train = tokenizer.texts_to_sequences(X_train)
```

```
X_test = tokenizer.texts_to_sequences(X_test)
```

Padding

In order for the neural network to work, the user must ensure that all sequences in a list have the same length. By default this is done by padding 0 in the beginning of each sequence until each sequence has the same length as the longest sequence. However, indicating the padding='post' argument, the padding will be done at the tail of the sequences.

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

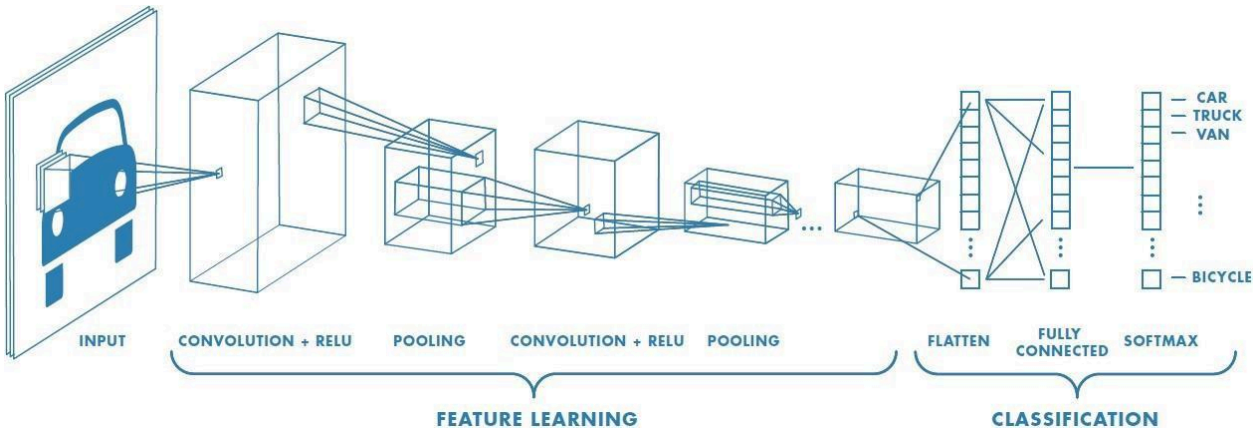
The data is now ready to be passed onto the model

4.3 Neural Network 1: Convolutional NN

4.3.1 Theory

Convolutional Neural Networks (CNN) differ from regular NN for its layers. The most significant is the convolutional layer, the core building block of a CNN. A convolution is a linear operation that involves the dot product of a set of weights with the input. The product is performed between an array of input data and a two-dimensional array of weights, which is called the filter. Then, the pooling takes place, and it Downsamples the input representation by taking the maximum value over a spatial window of a given size.

Finally, the network is flattened, which means it is converted into a one dimensional array, which will give the final solution in the output layer.



This kind of Neural Networks are frequently used in image processing, but is also very efficient at natural language processing (NLP), which is the task for this project. Usually it uses two-dimensional arrays, but since text is one-dimensional, a simplified version of the algorithm can be used.

4.3.2 Model creation

The first step in the model creation is the Sequential model, limited to single-input and single-ouptput stacks of layers.

```
model = Sequential()
```

Then, new layers are stacked on top of the model, such as the ones mentioned before in the theory section:

The embedding layer, whose function is to turn positive integers into dense vectors of fixed sized, is set as the first layer in a model. The dimensions of this array are defined by our vocabulary size. The input length is the length of the input sequences, mandatory when using flatten and dense layers, as we will.

```
model.add(Embedding(vocab_size, 32, input_length=50))
```

Hyperparameter tuning

The accuracy gets slightly improved (by 0.01) if the input length is 100 instead of 50, at the cost of a +1 minute execution time.

```
model.add(Embedding(vocab_size, 32, input_length=100))
```

Next, the convolutional layer, which produces a convolution kernel convolved with the input layer. We can also apply an activation function, in this case the ReLU (Rectified Linear Unit activation function). This linear function will output the input if it is positive, otherwise, it will output zero.

```
model.add(Conv1D(128, 5, activation='relu'))
```

A pooling layer can also be added, in this case, we will use the Global Average Pooling. Pooling consists in progressively reduce the size of the array to reduce the amount of total parameters, thus saving computational time.

```
model.add(GlobalAveragePooling1D())
```

A dense layer is a layer that is connected to its preceding layer with every neuron connected to each other. This is very common to get effective results, though it is logical to guess that it makes the execution to be notably slower. Only 1 dense layer wil be added, with the Hyperbolic Tangent activation function (tanh). It smoothly places all the values between -1 and 1. The sigmoid function is similar in this range, but has a different curve.

```
model.add(Dense(1, activation='tanh'))
```

The model is compiled using an optimizer, (adam is the most common one), a loss function (binary-crossentropy) and its metrics used will be accuracy.

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Finally, we can summarise all the layers added to the model:

```
print(model.summary())
```

```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_14 (Embedding)    (None, 100, 32)           799520

 conv1d_10 (Conv1D)          (None, 96, 128)           20608

 global_average_pooling1d_1   (None, 128)              0
 (GlobalAveragePooling1D)

 dense_15 (Dense)            (None, 1)                 129

=================================================================
Total params: 820,257
Trainable params: 820,257
Non-trainable params: 0
_________________________________________________________________
None
```

The model is now fitted a number of times, called epochs. Each time an output is produced, it is sent back to the input layer, making the process again, each time with a solution closer to perfection. However, there is always a point where the model does not improve significantly. It will be possible to see this in the epochs graph.

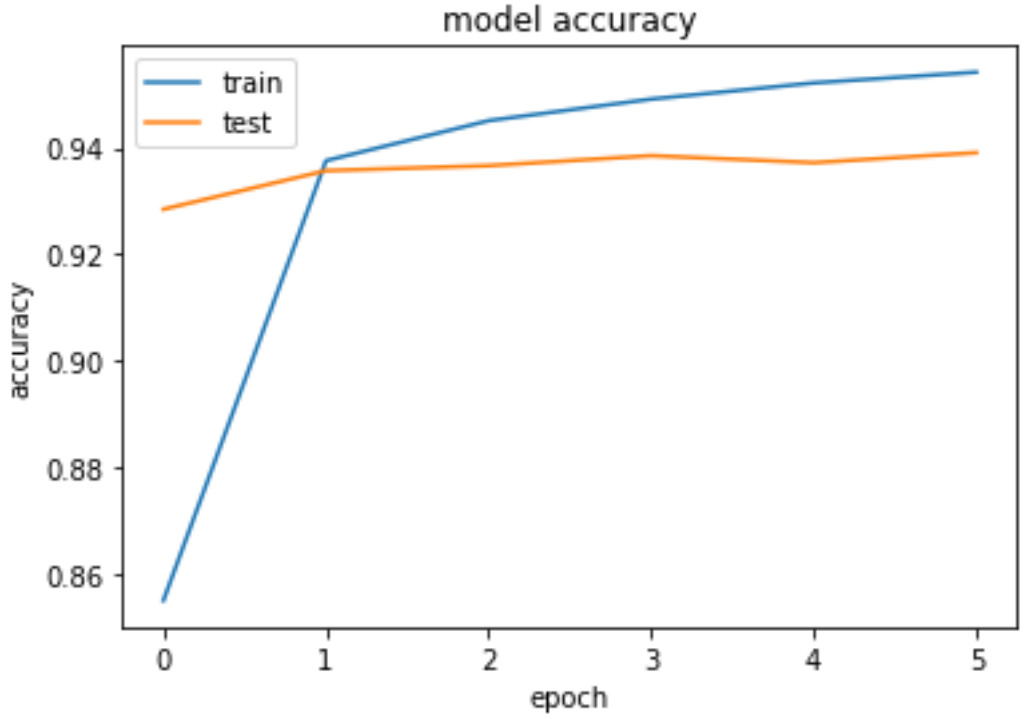
```
history = model.fit(X_train, y_train, batch_size=128, epochs=10, verbose=1,
validation_split=0.2)
```

As the last epochs take place, it is possible to see in the terminal how there is no valuable increase in the accuracy

```
Epoch 6/10
500/500 [==============================] - 25s 49ms/step -
loss: 0.2420 - accuracy: 0.9420 - val_loss: 0.3100 -
val_accuracy: 0.9323
Epoch 7/10
500/500 [==============================] - 24s 47ms/step -
loss: 0.2348 - accuracy: 0.9465 - val_loss: 0.3276 -
val_accuracy: 0.9343
Epoch 8/10
449/500 [=========================>....] - ETA: 2s - loss:
0.2282 - accuracy: 0.9488
```

4.3.3 Model evaluation

The model took a total of 3min 50s to fit and gave a final 93.53% accuracy. This data will be saved to compare it with the next Neural Network. As seen on the plot, the test accuracy hardly improves over epochs, so it is reasonable to limit the amount of epochs to 5 instead, which saves a considerable amount of time



4.4 Regular Neural Network

A Regular Neural Network works like the example explained in the Theory section, and has less complicated layers in its design. However, it does not mean that is necessarily worse than its fancy counterparts.

4.4.1 Model building

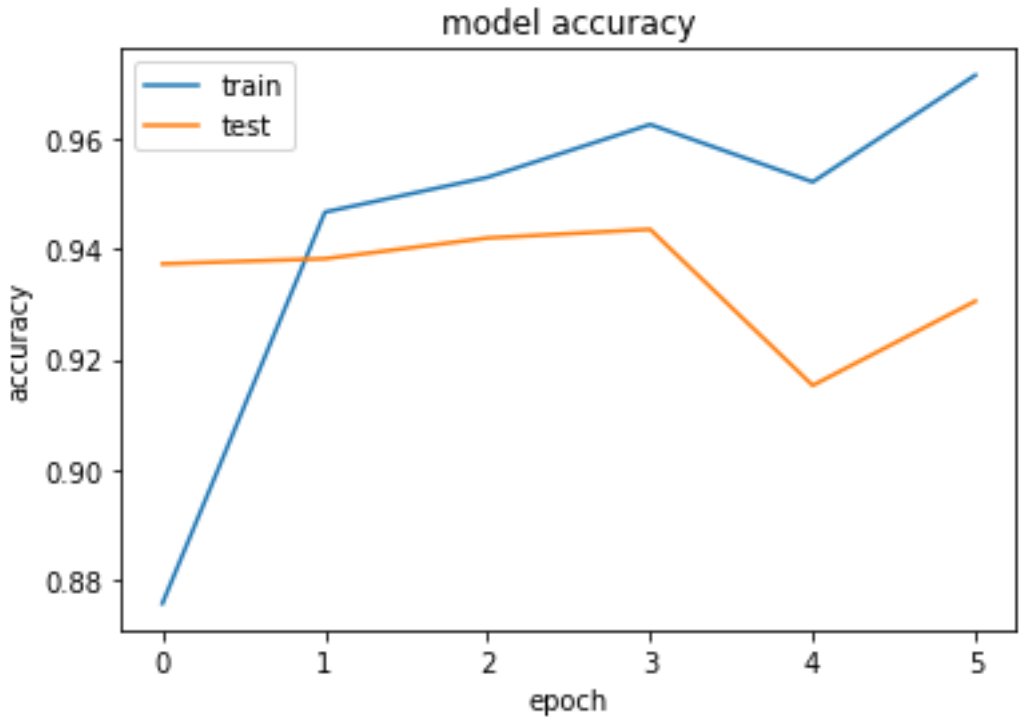
In order tu run a proper comparison, most of the layers will remain as similar to the previous model as possible. However, a dense layer with 128 units will be used to compensate for the lack of the convolutional one.

```
model = Sequential()
model.add(Embedding(vocab_size, 32, input_length=maxlen))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='tanh'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
print(model.summary())
```

```
Model: "sequential_18"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_18 (Embedding)    (None, 100, 32)           799520

 flatten_1 (Flatten)         (None, 3200)              0

 dense_19 (Dense)            (None, 128)               409728

 dense_20 (Dense)            (None, 1)                 129

=================================================================
Total params: 1,209,377
Trainable params: 1,209,377
Non-trainable params: 0
```

4.4.2 Model evaluating

This model took a significant smaller amount of time (1min 25s), and a similar accuracy (92.56%) but the results in the plot were rather surprising:



Sometimes, when training neural networks, drops in accuracy happen troughout epochs. model arrives close to some local maximum and then jumps off of it, since the learning rate is to high. This also affects the testing set in a more dramatic way, making the accuracy oscillate up and down throught the future epochs thus making it unnecessary to increase the epochs to increase accuracy.

4.5 Conclusions

Let us get a table comparing fit time and accuracy for both algorithms:

Accuracy and execution time of CNN and Regular NN		
	Fit time	**Accuracy**
Convolutional Neural Network	3min 50s	93.53%
Regular Neural Network	1min 25s	92.56%

Both methods provided a quite solid accuracy, specially when looking back at the traditional Machine Learning methods. They are also very much faster than the former, and require roughly the same amount of coding.

However, the theory behind Neural Networks is notably more complex than traditional Machine Learning algorithms, and the amount of combinations possible with layers and parameters are harder to grasp. Aditionally, since the majority of layers are hidden layers, it is hard to know what is going on inside of the network, difficulting the task of optimizing the algorithm. It works like a black box to the user, but at least, we were able to add our own layers with the functions we desired.

Although there is a comparison between these two networks, there are plenty more to choose, as well as infinite combination of layers possible in each model. There is also certain randomness in each model, so a decision should always be dutily considered by performing multiple combinations.

But if we were to choose a neural network to build from scratch, it would be recommendable to start with a Regular Neural Network first. Try out the most basic version, and from there, start scaling up with new layers, algorithms and tuning. There is few information known about how each model could take each minor change, so it is interesting to always check out what each new layer and modification does.

The world of Neural Networks if full of mysteries, but it is also the key to solve many mysteries than our biological neural networks cannot solve.

Bibliography

Plotly Dash library documentation. Retrieved January 10, 2022, from https://dash.plotly.com/
Plotly library documentation. Retrieved January 10, 2022, from https://plotly.com/python/
Keras library documentation. Retrieved January 12, 2022, from https://keras.io/api/

Dask library documentation. Retrieved January 12, 2022, from https://docs.dask.org/en/stable/

MongoDB documentation. Retrieved January 6, 2022, from https://docs.mongodb.com/manual/
Vu, D. V., (2019). *Python Word Clouds: How to Create a Word Cloud. DataCamp Community. Retrieved January 15, 2022,* from https://www.datacamp.com/community/tutorials/wordcloud-python
Manning, C. D., Raghavan, P., & Schütze, H. (2018). *Introduction to information retrieval*. Cambridge University Press. https://nlp.stanford.edu/IR-book/