Hogeschool van Amsterdam

Author
Jaime Raynaud – 500884682
BD3

School
Zarina Efendieva & Evert-Jan
Hogeschool van Amsterdam

Date
9/1/2022

Report Data Engineer and Data Scientist individual assignment 2

Sentiment Analysis on Hotel Reviews

Table of contents

1. Summary

Imagine that you are thinking of going on vacation the next summer, one of the multiple tasks that you have to do before departure is to look for a hotel where to stay. If you enter a site like Booking or TripAdvisor chances are that one of the main reasons to choose a hotel is their reviews. So, nowadays hotels can benefit from all this information that their guests publish on the multiple hotel recommendations sites. Reviews can tell you how the hotel is satisfying the customer's needs, which is crucial for developing marketing strategies.

With the grown of Big Data, tools like Sentiment Analysis can help businesses understand their customers. Sentiment analysis is contextual mining of text which identifies and extracts subjective information in source material, and helps a business to understand the social sentiment of their brand, product or service while monitoring online conversations.

In this second report, we will see how using the knowledge acquired during the course Big Data Scientist and Engineer blok 2 and trying completely new different techniques, compared with the first assignment, we can predict whether or not a review is positive or negative, speeding up the process thanks to Dask or predicting with a complex Recurrent Neural Network.

At the end of it, we will see how this forecasting task is satisfied with the Deep Learning techniques that I will use, achieving accuracy in the predictions superior to 92%. Also, we will visualize the data with a living connection to a MongoDB database, in a visualization that follows the Mantra of visualization principles.

2. Introduction

This report will continue the work done in the first Individual Assignment presenting different techniques, this time I will use a MongoDB database, apply parallel computing with Dask to Machine Learning algorithms and create a Neural network to compare results with the ones achieved in the first assignment. Moreover, a Dashboard will be presented to visualize the used dataset, programmed with the Dash Python library.

To carry out this task I will start describing the use of the MongoDB database.

Secondly, the use of parallel computing with Dask will be described, and the results will be compared with the ones obtained in the first assignment.

In third place, I will talk about the two Neural networks created, explaining their differences and comparing their results in the forecasting task.

Finally, I will show the Dashboard that I created with Dash, explaining how it satisfies the Mantras of visualization and its multiple tabs.

To be able to carry out this task we will use the Visual Studio environment, programming in Python using useful libraries for Data Science like Pandas, Scikit-Learn, Dask, and Dash. Also, to store the data in a database I will use MongoDB database.

3. Methods

In this section, we will see the different requirements asked for this assignment, from the application of the MongoDB database, use of Dask, the Neural Networks created, and the Dashboard.

3.1. MongoDB

MongoDB is a document-oriented NoSQL database used for high volume data storage. Instead of using tables and rows as in the traditional relational databases, MongoDB makes use of collections and documents. Documents consist of key-value pairs which are the basic unit of data in MongoDB. Collections contain sets of documents and function which is the equivalent of relational database tables. MongoDB is a database which came into light around the mid-2000s. (Taylor, 2022)

MongoDB has a series of features that make it different from other databases:

1. Each database contains collections which in turn contains documents. Each document can be different with a varying number of fields. The size and content of each document can be different from each other.
2. The document structure is more in line with how developers construct their classes and objects in their respective programming languages. Developers will often say that their classes are not rows and columns but have a clear structure with key-value pairs.
3. The rows (or documents as called in MongoDB) doesn't need to have a schema defined beforehand. Instead, the fields can be created on the fly.
4. The data model available within MongoDB allows you to represent hierarchical relationships, to store arrays, and other more complex structures more easily.
5. Scalability – The MongoDB environments are very scalable. Companies across the world have defined clusters with some of them running 100+ nodes with around millions of documents within the database.

(Taylor, 2022)

Once we have seen the characteristics of this database let's see the application in the assignment.

First of all, I will talk about the database that I have created. Its name is *assignment2*, containing 3 different collections, one for the Kaggle dataset, without modifications, another for the models, where I am storing the stemmed and cleaned reviews with their labels, and one final collection where I store data from the different hotels appearing in the Kaggle dataset. In this final dataset, I store the hotel names, scores, address, total number of reviews, latitude, longitude, city, and country. All this data will be very useful for the data visualization part.

Now that the data stored in the database and the theory behind MongoDB is explained it is time to show the different Python functions that interact with it.

The first one is the *get_all(data)* function. With this function, I can get any of the collections that I have stored in the database, so I can get the data directly from it when using it in the visualization.

```
def get_all(data):
        print('\nCreating database connection...')
        client = MongoClient("localhost:27017")
        db = client["assignment2"]
        collection = db[data]
        print('\nGetting data...')
        return pd.DataFrame(list(collection.find({})))
```

Secondly, I needed a function to upload the data to the database, so I created the function *upload_data(df).* With this function I can upload any pandas dataframe to the database, specifying the name of the collection to upload it in the function.

```
def upload_data(df):
    print('\nCreating database connection...')
    conn = MongoClient("localhost:27017")
    db = conn.assignment2
    collection = db.hotel_reviews
    print('\nTrying to insert the data...')
    collection.insert_many(df.to_dict('records'))
    print('\nSuccessful uploaded data')
```

Finally, a live connection to filter data during the process of running the script is implemented. Meaning during the process of filtering and plotting new data is transferred from the database to the python dashboard.

This is done by the function aggregate_fun(). With this function, I am using a MongoDB aggregate function to get the data from those hotels whose total number of reviews is superior to 5000.

```
# Bubble graph
# Applying aggregation function in MongoDB (instead of query):
data_triple = db.aggregate_fun()

triple = px.scatter(data_triple, x="Additional_Number_of_Scoring", y="Average_Score", size="Total_Number_of_Reviews", color="Hotel_Name",
        hover_name="Hotel_Name", size_max=60, width=920, title = 'Hotels scores and number of reviews')
```

This data is used in the bubble scatter plot in the dashboard, so just the most reviewed hotels are visualized, to not have thousands of hotels as small bubbles in the visualization.

```
def aggregate_fun():
    print('\nCreating database connection...')
    client = MongoClient("localhost:27017")
    db = client["assignment2"]

    print('\nApplying aggregate function...')
    pipeline = [
    {
        "$match": { 'Total_Number_of_Reviews': { '$gt': 5000 } }
    },
    ]
    return pd.DataFrame(db.data_hotels.aggregate(pipeline))
```

Now that all the database-related processes are explained, I will continue talking about the parallel computing implementation with Dask.

3.2. Parallel computing with Dask

Dask is popularly known as a ‘parallel computing’ python library that has been designed to run across multiple systems.

Dask can efficiently perform parallel computations on a single machine using multi-core CPUs. For example, if you have a quad-core processor, Dask can effectively use all 4 cores of your system simultaneously for processing. In order to use lesser memory during computations, Dask stores the complete data on the disk, and uses chunks of data (smaller parts, rather than the whole data) from the disk for processing. During the processing, the intermediate values generated (if any) are discarded as soon as possible, to save memory consumption. (Singh, 2020)

In summary, Dask can run on a cluster of machines to process data efficiently as it uses all the cores of the connected machines. One interesting fact here is that it is not necessary that all machines should have the same number of cores. If one system has 2 cores while the other has 4 cores, Dask can handle these variations internally. (Singh, 2020)

Dask supports the Pandas dataframe, Numpy array data structures and Scikit-learn to analyze large datasets. Basically, Dask lets you scale pandas, numpy, and Scikit-learn between others, with minimum changes in our code format. (Singh, 2020)

Dask ML provides scalable machine learning algorithms in python which are compatible with scikit-learn. Let us first understand how scikit-learn handles the computations and then we will look at how Dask performs these operations differently.

A user can perform parallel computing using scikit-learn (on a single machine) by setting the parameter njobs = -1. Scikit-learn uses Joblib to perform these parallel computations. Joblib is a library in python that provides support for parallelization. When you call the .fit() function, based on the tasks to be performed (whether it is a hyperparameter search or fitting a model), Joblib distributes the task over the available cores. To understand Joblib in detail, you can have a look at this documentation. (Singh, 2020)

Even though parallel computations can be performed using scikit-learn, it cannot be scaled to multiple machines. On the other hand, Dask works well on a single machine and can also be scaled up to a cluster of machines. (Singh, 2020)

Dask has a central task scheduler and a set of workers. The scheduler assigns tasks to the workers. Each worker is assigned a number of cores on which it can perform computations. The workers provide two functions:

- compute tasks as assigned by the scheduler
- serve results to other workers on demand

Below is an example that explains how a conversation between a scheduler and workers looks like (this has been given by one of the developers of Dask, Matthew Rocklin):

The central task scheduler sends jobs (python functions) to lots of worker processes, either on the same machine or on a cluster:

Worker A, please compute x = f(1), Worker B please compute y = g(2)

Worker A, when g(2) is done please get y from Worker B and compute z = h(x, y)

This should give us a clear idea of how Dask works. Now we will discuss machine learning models and Dask-search CV. (Singh, 2020)

As we have seen previously, sklearn provides parallel computing (on a single CPU) using Joblib. In order to parallelize multiple sklearn estimators, we can directly use Dask by adding a few lines of code (without having to make modifications in the existing code). (Singh, 2020)

3.2.1. Dask Logistic regression

The first use of Dask that we will discuss is just its implementation as the machine learning model to use. For simple machine learning algorithms which use Numpy arrays, Dask ML re-implements these algorithms. Dask replaces numpy arrays with Dask arrays to achieve scalable algorithms.

```
from dask_ml.linear_model import LogisticRegression as DaskLogisticRegression
from sklearn.linear_model import LogisticRegression
```

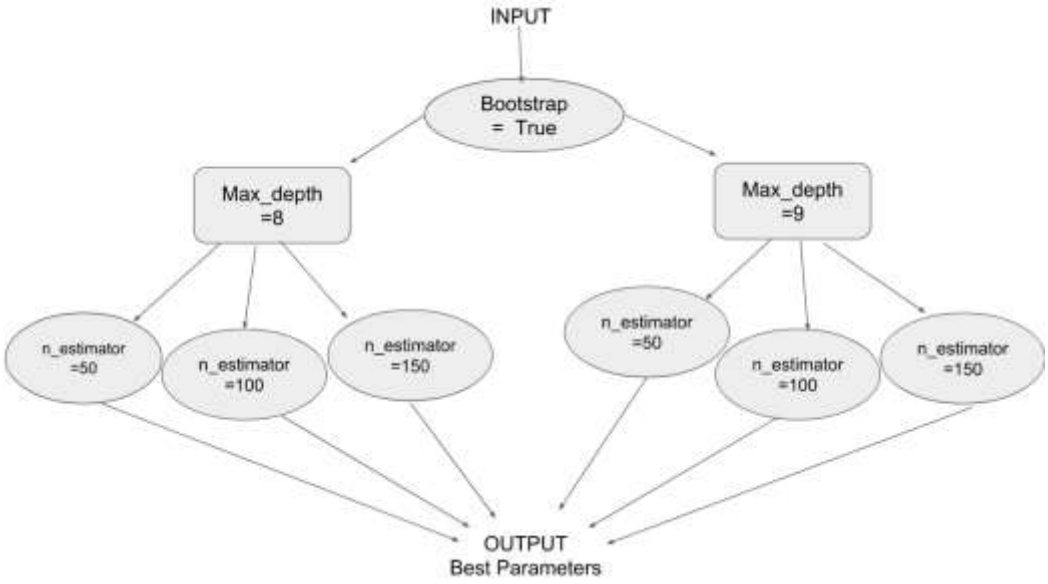
3.2.2. Dask GridSearchCV

Hyperparameter tuning is an important step in model building and can greatly affect the performance of our model. Machine learning models have multiple hyperparameters and it is not easy to figure out which parameter would work best for a particular case. Performing this task manually is generally a tedious process. In order to simplify the process, sklearn provides Gridsearch for hyperparameter tuning. The user is required to give the values for parameters and Gridsearch gives you the best combination of these parameters. (Singh, 2020)

sklearn Gridsearch: For each combination of the parameters, sklearn Gridsearch executes the tasks, sometimes ending up repeating a single task multiple times. As you can see from the below graph, this is not exactly the most efficient method:

Dask-Search CV: Parallel to Gridsearch CV in sklearn, Dask provides a library called Dask-search CV (Dask-search CV is now included in Dask ML). It merges steps so that there are less repetitions. (Singh, 2020)

The following graph explains the working of Dask-Search CV:



Now lets talk about the implementation of Dask in my code.

```
def dask_model(df):
    df = df.head(10000)
    print('\n')
    print('Applying tfidf...')
    print('\n')

    X = tfidf(df)
    y = df.label

    from sklearn import decomposition
    from sklearn.pipeline import Pipeline

    print('\n')
    print('Applying models without Dask...')
    print('\n')

    t0 = time.time()
    logistic = LogisticRegression()
    pca = decomposition.PCA()
    pipe = Pipeline(steps=[('pca', pca),
                           ('logistic', logistic)])

    grid = dict(pca__n_components=[50, 100, 250],
            logistic__C=[1e-4, 1.0, 1e4],
            logistic__penalty=['l2'])

    estimator = GridSearchCV(pipe, grid, n_jobs=-1)
    estimator.fit(X, y)
    t1 = time.time()
    print('\n')
    print('Best parameters without Dask: ',estimator.best_params_)
    print('Best score without Dask: ', estimator.best_score_)
    print("Time to process without Dask {}".format(t1-t0))
    print('\n')

    print('\n')
    print('Applying models with Dask...')
    print('\n')

    t0 = time.time()
    logistic = DaskLogisticRegression()
    pca = decomposition.PCA()
    pipe = Pipeline(steps=[('pca', pca),
                           ('logistic', logistic)])

    grid = dict(pca__n_components=[50, 100, 250],
            logistic__C=[1e-4, 1.0, 1e4],
            logistic__penalty=['l2'])

    destimator = DaskGridSearchCV(pipe, grid)
    destimator.fit(X, y)
    t1 = time.time()
    print('\n')
    print('Best parameters: ',destimator.best_params_)
    print('Best score with Dask: ', destimator.best_score_)
    print("Time to process with Dask {}".format(t1-t0))
```

In the function shown, Dask is applied during the use of GridSearchCV and in the Logistic Regression model. I wanted to compare the results and times of execution of a model with scikit learn grid search and logistic regression versus a model with dask grid search and dask logistic regression.

To show the results below I have used 50000 rows of data to speed up the calculations and because I had a RAM error when executing with bigger amounts of data.

	Time (s)	*Score*	*Best parameters*
Scikit Learn	119.83	0.91578	C: 1, penalty: l2, Pca: 250
Dask	41.99	0.91544	C: 10000, penalty: l2, Pca: 250

So, after all, we can see that applying Dask considerably speeds up the calculations, being 285% faster in this case. Although the accuracy of the model has been slightly decreased, we can conclude that applying Dask to our algorithms is worth it.

3.3. Neural Networks

Another of the objectives of this assignment was to implement two different types of Neural Networks, to see if they can perform better than the rest of the classical machine learning techniques that we have implemented in this assignment and the previous one. But first, let's see some theory about Neural networks.

Neural networks are a means of doing machine learning, in which a computer learns to perform some task by analyzing training examples. Usually, the examples have been hand-labeled in advance. An object recognition system, for instance, might be fed thousands of labeled images of cars, houses, coffee cups, and so on, and it would find visual patterns in the images that consistently correlate with particular labels. (*Explained: Neural Networks*, 2017)

Modeled loosely on the human brain, a neural net consists of thousands or even millions of simple processing nodes that are densely interconnected. Most of today's neural nets are organized into layers of nodes, and they're "feed-forward," meaning that data moves through them in only one direction. An individual node might be connected to several nodes in the layer beneath it, from which it receives data, and several nodes in the layer above it, to which it sends data. (*Explained: Neural Networks*, 2017)

To each of its incoming connections, a node will assign a number known as a "weight." When the network is active, the node receives a different data item — a different number — over each of its connections and multiplies it by the associated weight. It

then adds the resulting products together, yielding a single number. If that number is below a threshold value, the node passes no data to the next layer. If the number exceeds the threshold value, the node “fires,” which in today’s neural nets generally means sending the number — the sum of the weighted inputs — along all its outgoing connections. (*Explained: Neural Networks*, 2017)

When a neural net is being trained, all of its weights and thresholds are initially set to random values. Training data is fed to the bottom layer — the input layer — and it passes through the succeeding layers, getting multiplied and added together in complex ways, until it finally arrives, radically transformed, at the output layer. During training, the weights and thresholds are continually adjusted until training data with the same labels consistently yield similar outputs. (*Explained: Neural Networks*, 2017)

For this assignment, in particular, my goal is to compare a Convolutional Neural net with a Recurrent Neural net. Because of this, let’s explain the characteristics of each and their differences.

3.3.1. Differences between CNNs and RNNs

The main difference between CNN and RNN is the ability to process temporal information or data that comes in sequences, such as a sentence for example. Moreover, convolutional neural networks and recurrent neural networks are used for completely different purposes, and there are differences in the structures of the neural networks themselves to fit those different use cases.

CNNs employ filters within convolutional layers to transform data. Whereas, RNNs reuse activation functions from other data points in the sequence to generate the next output in a series.

While it is a frequently asked question, once you look at the structure of both neural networks and understand what they are used for, the difference between CNN and RNN will become clear. (TELUS International, 2015)

3.3.2. Convolutional Neural Networks

Convolutional neural networks are one of the most common types of neural networks used in computer vision to recognize objects and patterns in images. One of their defining traits is the use of filters within convolutional layers.

CNNs have unique layers called convolutional layers which separate them from RNNs and other neural networks. Within a convolutional layer, the input is transformed before being passed to the next layer. A CNN transforms the data by using filters. A filter in a CNN is simply a matrix of randomized number values like in the diagram below.

0.230	0.380	0.971
0.402	0.119	0.886
0.693	0.563	0.771

But, where do CNNs fall short? CNNs are great at interpreting visual data and data that does not come in a sequence. However, they fail in interpreting temporal information such as videos (which are essentially a sequence of individual images) and blocks of text.

Entity extraction in text is a great example of how data in different parts of a sequence can affect each other. With entities, the words that come before and after the entity in the sentence have a direct effect on how they are classified. In order to deal with temporal or sequential data, like sentences, we have to use algorithms that are designed to learn from past data and 'future data' in the sequence. Luckily, recurrent neural networks do just that. (TELUS International, 2015)

3.3.3. Recurrent Neural Networks

Recurrent neural networks are networks that are designed to interpret temporal or sequential information. RNNs use other data points in a sequence to make better predictions. They do this by taking in input and reusing the activations of previous nodes or later nodes in the sequence to influence the output. As mentioned previously, this is important in tasks like entity extraction. Take, for example, the following text: President Roosevelt was one of the most influential presidents in American history. However, Roosevelt Street in Manhattan was not named after him.

In the first sentence, Roosevelt should be labeled as a person entity. Whereas, in the second sentence it should be labeled as a street name or a location. Knowing these distinctions is not possible without taking into account the words before them, "president," and after them, "street".

This is why Recurrent Neural Networks are the perfect ones for processing texts, from autocorrection tasks to sentiment analysis, that will be this case use.

Today, data scientists use RNNs to do much more incredible things. From generating text and captions for images to creating music and predicting stock market fluctuations, RNNs have endless potential use cases. (TELUS International, 2015)

The main purpose of this part of the assignment is to compare the results obtained with both types of networks, so now, let's talk about the two NNs created and the tweaking of their layers to obtain better results.

3.3.4. Tunning of the Neural Networks

To begin with, I took the two models uploaded to the DLO, in the script *02_KERAS_NN_CNN_RNN_SentimentAnalysis_IMDB.py*.

In this script, you can find the two Neural Networks shown below. With these two without modifications, I obtained good results, but the objective of this part of the assignment is to see if I can improve these results by changing the batch size, number of epochs, and the configuration of layers, by adding or modifying the existing ones.

```
# create the model
model = Sequential()
model.add(Embedding(vocab_size, 32, input_length=maxlen))
    #  text is 1 dimensional, so Conv1D
    #  the kernel in this case is a vector of length 5, not a 2 dimensional matrix
model.add(Conv1D(128, 5, activation='relu'))
    #  the pooling layer in this case is also 1 dimensional
model.add(GlobalMaxPooling1D())
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
```

First Convolutional Neural Network

```
# create the model
model = Sequential()
model.add(Embedding(vocab_size, 32, input_length=maxlen))
model.add(LSTM(128))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
```

First Recurrent Neural Network

The first, tweaking was modifying the batch size, and number of epochs. The batch size corresponds to the number of examples used in one iteration, while the number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset.

Let's show a table with the different combinations of values that I tried and the results obtained:

Epochs	3	6	6	6	9
Batch size	256	256	320	64	256
Accuracy	0.907	**0.9135**	0.906	0.703	0.904

Take into account that the accuracy values are for the Recurrent Neural Network.

As we can see the best combination is to use a batch size of 256 and 6 epochs, so this is the one that we will use when tweaking the layers of the NNs.

Starting with the CNN, I decided to take advice on the article created by Brownlee (2020b) in which he uses a certain configuration of layers, doing the next mods:

1. Adding an Embedding layer
2. Modifying the Conv1D layer, with an initial value of 128 for the filters parameter, after I changed to 64, obtaining better results.
3. Changing the Global pooling one, for a MaxPooling1D layer, as Brownlee (2020b) uses.
4. Adding a Flatten layer
5. Adding a Dense layer just before the output layer, with an initial number of units of 32, I also tried with 10, obtaining better results.

```
model = Sequential()
model.add(Embedding(vocab_size, 32, input_length=maxlen))
model.add(Conv1D(filters=64, kernel_size=5, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
```

Let's see in a table the steps taken and their accuracies:

Initial CNN	**CNN with new layers (MaxPooling1D, Flatten and Dense)**	**CNN with a 32 units in the Dense layer**	**Final CNN with 64 filters in the Conv1D layer**
0.9135	0.915	0.913	**0.916**

So the highest accuracy obtained was 0.916 in the validation set. With this, we have tweaked the CNN and obtained a better version than the original one, let's try now with the RNN.

For the RNN I took as a model the one that I constructed for the Datacamp course on Sentiment analysis with RNNs.

```
model = Sequential()
model.add(Embedding(vocab_size, 32, input_length=maxlen))
model.add(Dense(32, activation='relu', name='Dense1'))
model.add(Dropout(rate = 0.25))
model.add(LSTM(64, return_sequences=True, dropout=0.15, name='LSTM'))
model.add(GRU(64, return_sequences=False, dropout=0.15, name='GRU'))
model.add(Dense(64, name='Dense2'))
model.add(Dropout(rate = 0.25))
model.add(Dense(32, name='Dense3'))
model.add(Dense(1, activation='sigmoid', name = 'Output'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
```

For this reason, I tried in order:

1. Adding a trainable=True parameter to the Embedding layer, which dropped the accuracy.
2. Adding the layer Dense1, which initially also dropped the accuracy, but at the end helped rising it.
3. Adding a Dropout layer, to remove noise.
4. Modifying the LSTM layer, changing the number of units to 64, adding the return_sequences=True and a dropout of 0.15.
5. Adding a GRU with the shown parameters, to avoid vanishing or exploding gradient, as seen during the Datacamp course (*RNNs*, n.d.).

After this, the accuracy of the model from 0.9135 before mods to 0.917.

The second round of modifications is as follows:

1. Adding a new Dense2 layer.
2. Another Dropout layer.
3. A final Dense3 layer just before the output layer.

The accuracy was raised to 0.9185, after this I tried changing the number of units of Dense3 from 64 to 32, obtaining a new record: 0.921 of accuracy on the validation set.

Initial RNN	**RNN with the first round of mods**	**RNN with the second round of mods**	**Final RNN with 32 units in the Dense3 layer**
0.9135	0.917	0.9185	**0.921**

Finally, we have tweaked both NNs and obtained better scores than the initial ones, seeing that, as the theory suggests, RNNs perform better for the Sentiment Analysis task.

3.4. Dashboard

For building my Dashboard I will use the Python library Dash. Dash is an open-source framework for building data visualization interfaces. It will help us build our Dashboard to visualize the data used during this project.

Why do we need visualization?

Firstly the need for any kind of data or information visualization is to gain insights into the data and not just pictures or graphs. Moreover, visualization allows us to detect faulty or missing data or inconsistencies within the data.

Anomalies can also be found and corrected. Visualization is multidisciplinary, that is any person can use it to get a deeper understanding of data. It can help convert raw insights into business problems. Colors add different perceptions and can help us think differently. Visualization should penetrate the visual domain of an individual.

Too much visualization can cause problems and staggering of data. It can also hide important insights. For this purpose, we need fixed guidelines on how to apply visualizations and what to show, and what to hide.

Ben Shneiderman's Visual Information Seeking Mantra

In his paper titled "The eyes have it: a task by data type taxonomy for information visualizations", Ben Schneiderman suggested 7 principles for visualization of data, lets see some of them:

First principle: Overview

The first step of the Holy Trinity of Visual Information Seeking is 'Overview'. In this step, it is recommended to visualize the given data set in a simple way without going into too much detail. We do not want to lose any information in this step, thus it is recommended to not apply any filters and just visualize the given data to get an idea about it.

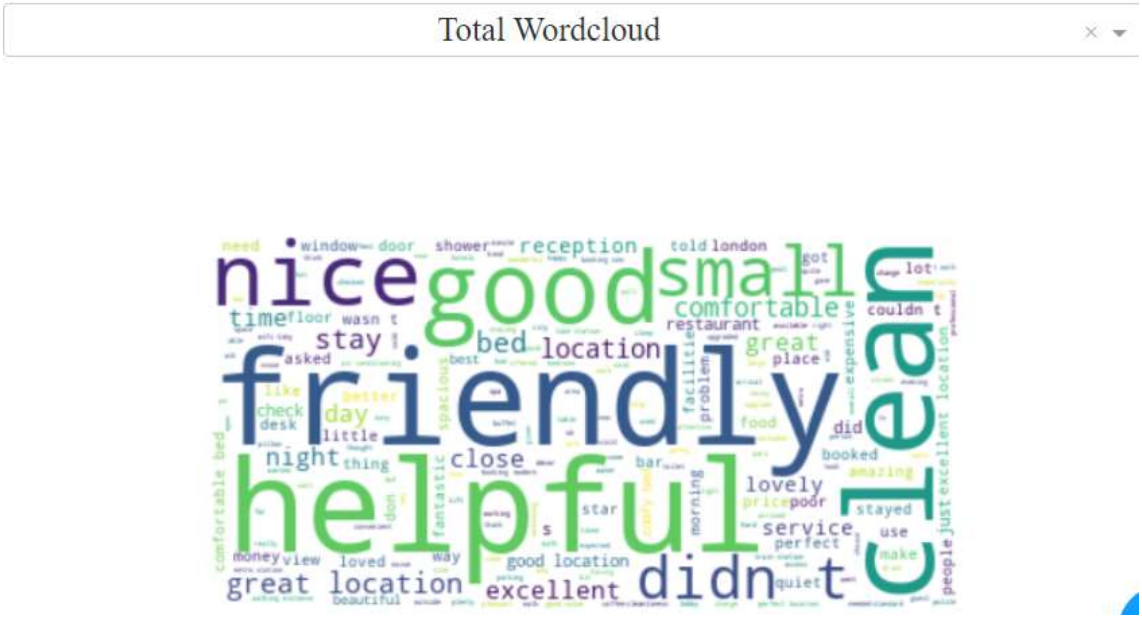
As the first step, we will visualize the Hotels data set using a table, a scatter plot graph, and a word cloud.

	Name	Average sc	Number of r	Country	City	Adress	Additional
	Britannia Inter	7.1	9086	United Kingdom	London	163 Marsh Wall	2682
	Park Plaza West	8.7	12158	United Kingdom	London	Westminster Bri	2623
	Strand Palace H	8.1	9568	United Kingdom	City of Westmir	372 Strand West	2288
	DoubleTree by H	8.7	7491	United Kingdom		7 Pepys Street	1936
	Copthorne Tara	8.1	7105	United Kingdom	London	Scarsdale Place	1831
	Hilton London M	7.5	6977	United Kingdom	London	225 Edgware Roa	1485
	M by Montcalm S	9.1	4802	United Kingdom	London	151 157 City Rc	1471
	Millennium Glo	7.8	5726	United Kingdom	London	4 18 Harringtor	1444
	Hilton London H	8.8	4305	United Kingdom	London	Lakeside Way Br	1427
	Park Plaza Cour	8.4	6117	United Kingdom	London	1 Addington Str	1322

1 / 150

Table

Scatter plot



Wordcloud

With the table, we have all the relevant summarized information about the Hotels in the dataset, gaining a general overview of it.

On the other hand with the scatter plot we can quickly visualize which ones are the most important hotels, visualizing both scores and the number of reviews of the different hotels in the dataset.

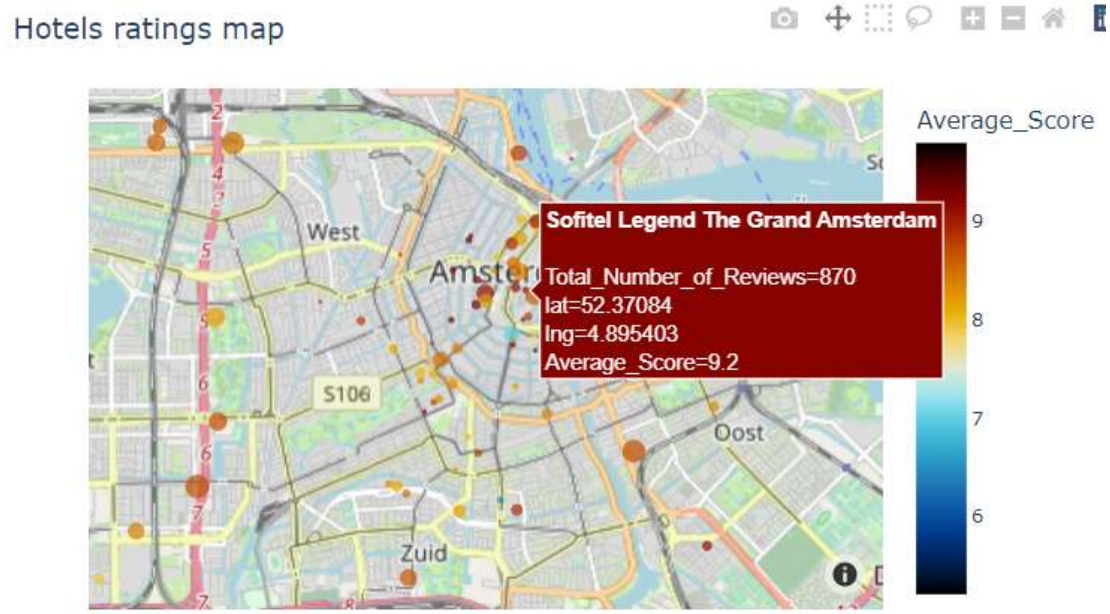
Finally, with the word cloud we have a summary of the most used words in the reviews of the dataset.

Second and third: Zoom and filter

After understanding the basic distribution of the data set in the first step, the next steps that follow are ‘Zoom’ and ‘Filter’. Here, we proceed to focus on the part of the data set that is of value to us. Zooming and Filtering help highlight the objects of interest using various techniques.

There are two ways we can zoom in on the data:

- On the map, we can apply zoom to see the different hotels all around Europe in the dataset. It allows us to see different cities where the hotels are present like Amsterdam or Paris, so we can see where the hotels are located and the score that they have associated with colors, to help us decide our holidays choice.



- We can also Zoom in the scatter plot, to see closely which hotels are in certain ranges of scores, for example, looking between an Average score of 7.5 and additional score of 2000 we have the following hotels:

We can apply filters on the table of our Dashboard, here we can filter between all the existent categories. For example, let's obtain the hotels that have more than an 8 in the average score, more than 8000 reviews, and are in Milano:

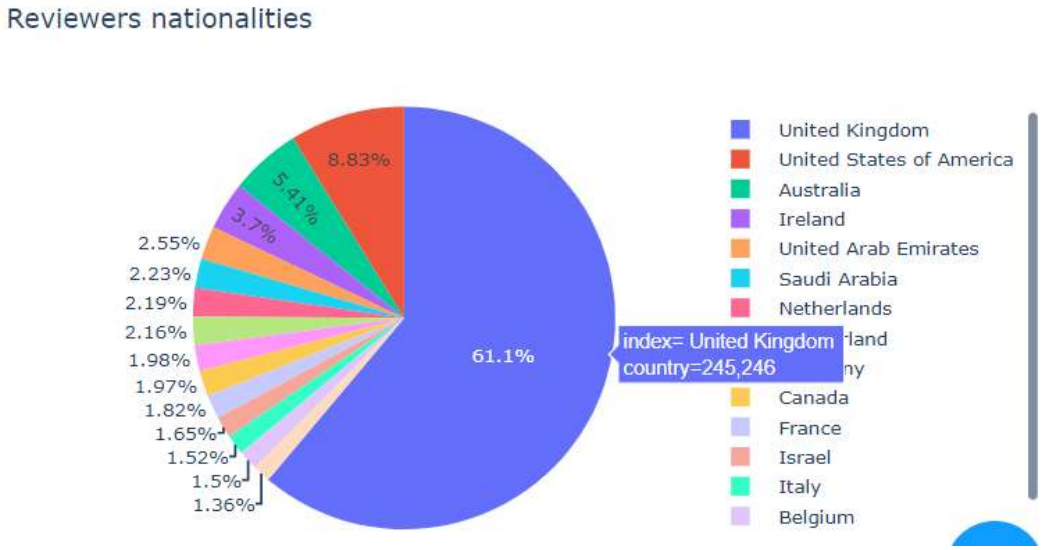
	Name	Average sc	Number of r	Country	City	Adress	Additional
					Milano		
	Hotel Da Vinci	8.1	16670	Italia	Milano	Via Senigallia 904	4
	Hotel degli Ar	8.3	10842	Italia	Milano	Viale Sarca 336563	4

Details on demand

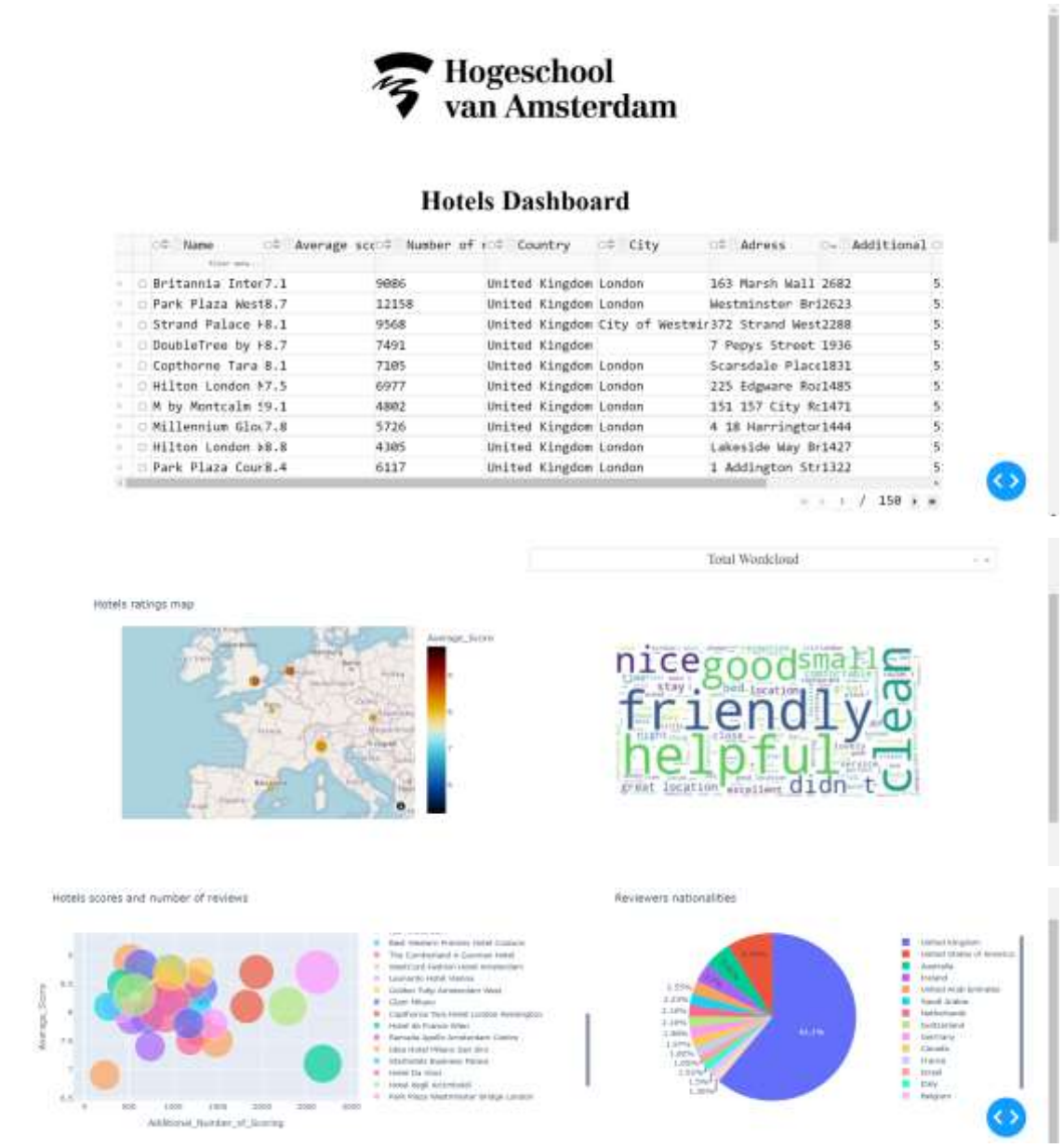
We identify areas of interest in the first step (Overview), and dig deeper into the data using zooming and filtering in the second step, the next step naturally should be to find exact details which will help us find interesting facts from the data set. The final step of the Holy Trinity is details on demand. Here we use the full power of interactivity that are provided by the libraries that we use.

For example, thanks to the library ‘plotly’ we can make the graphs that we have shown with a high level of interactivity. When we hover our cursor above each point in the map, the scatter plot, or the pie chart, we get the details for that particular hotel or nationality of reviewers.

In the pie chart for example you can see the number of reviewers per country, by hoving over the parts of the pie.



Finally, the Dashboard looks like this:

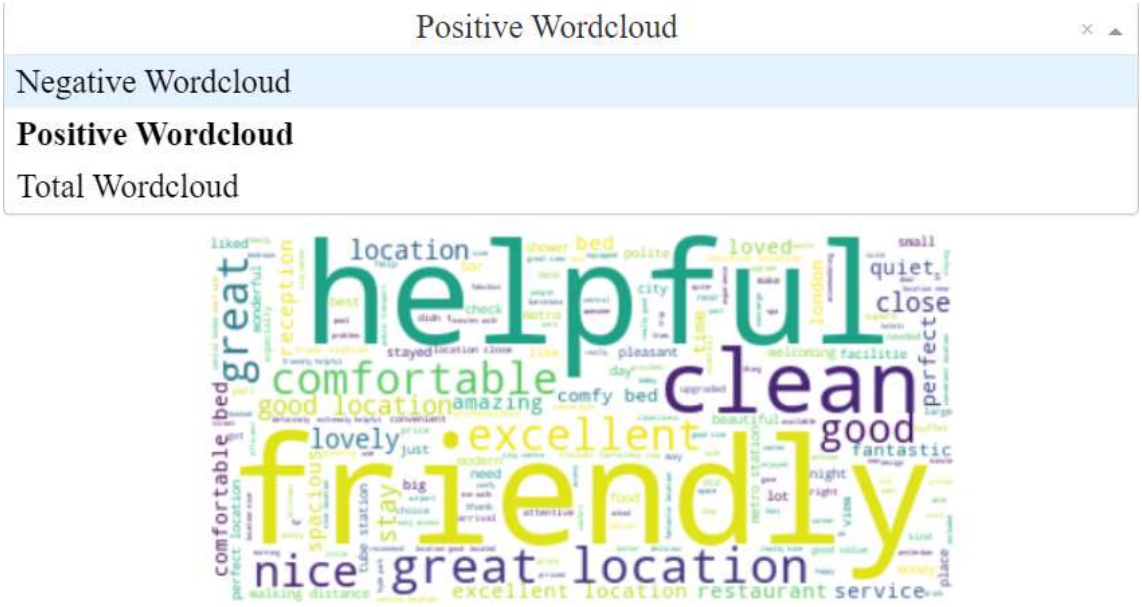


Now let's talk about the visualizations selected.

Firstly the table gives us a general overview of the dataset, allowing us to filter and order the data easily.

Secondly, the map is a very intuitive and classical way of looking for hotels in a specific area, allowing us to zoom and see details like the scores in colors of different hotels.

Then we have the word cloud. I decided to include three different word clouds that the user can select with the bar above since they give us a general overview of what the reviews are talking about. A general word cloud of all the reviews, one for the positive reviews, and one for the negatives.



We have also the scatter plot that is an appropriate way of visualizing multiple important features with the axis and the size of the bubbles, in this case, the scores and the number of reviews, since they are the most important when looking for a hotel.

Finally, I believed that it was also important to gain some insight about the reviewers, that is why I chose to represent the most common nationalities of the people that have visited the hotels.

4. Conclusion and Recommendations

We have come to the final of this report on sentiment analysis in hotel reviews. We have been seen all the steps involved in this new assignment. I started storing all the data in a new MongoDB database, creating functions to upload and obtain the data, as well as an aggregate function with a live connection with the visualization.

After that, we saw the applications of Dask, comparing its accuracy and time of execution with a Scikit learn GridSearchCV and Logistic Regression model. Improving the time of execution a 285%.

Then, we saw a full application of two different Neural networks, a Convolutional and a Recursive. Proving that the theory is true, obtaining an accuracy of 0.921. To obtain this accuracy, I tweaked the number of epochs, batch size and add different layers to both NNs, so they will perform better.

Now, and to finish this report, I will make some recommendations that will help future versions of this project improve the results here obtained.

- Try Dask on more Machine Learning algorithms, to see if they speed up considerably, like XGBoost.

- Use even more configurations of the layers or add new ones, usually with more complexity of the layers the results will improve.

- Add more visual objects to the visualization to gain even more insight, and add CSS to it.

5. References

Brownlee, J. (2020a, September 3). *Deep Convolutional Neural Network for Sentiment Analysis (Text Classification)*. Machine Learning Mastery. https://machinelearningmastery.com/develop-word-embedding-model-predicting-movie-review-sentiment/

Brownlee, J. (2020b, September 3). *Deep Convolutional Neural Network for Sentiment Analysis (Text Classification)*. Machine Learning Mastery. https://machinelearningmastery.com/develop-word-embedding-model-predicting-movie-review-sentiment/

Explained: Neural networks. (2017, April 14). MIT News | Massachusetts Institute of Technology. https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414

Real Python. (2020, December 17). *Develop Data Visualization Interfaces in Python With Dash*. Dash. https://realpython.com/python-dash/

RNN Architecture - Vanishing and exploding gradients. (2021). DataCamp. https://campus.datacamp.com/courses/recurrent-neural-networks-for-language-modeling-in-python/rnn-architecture

Singh, A. (2020, May 31). *DASK | Handling Big Datasets For Machine Learning Using Dask*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2018/08/dask-big-datasets-machine_learning-python/#:%7E:text=In%20simple%20words%2C%20Dask%20arrays,larger%20than%20the%20memory%20size.

Taylor, D. (2022, January 1). *What is MongoDB? Introduction, Architecture, Features & Example*. Guru99. https://www.guru99.com/what-is-mongodb.html

TELUS International. (2015, February 5). *What Is The Difference Between CNN And RNN?* https://www.telusinternational.com/articles/difference-between-cnn-and-rnn