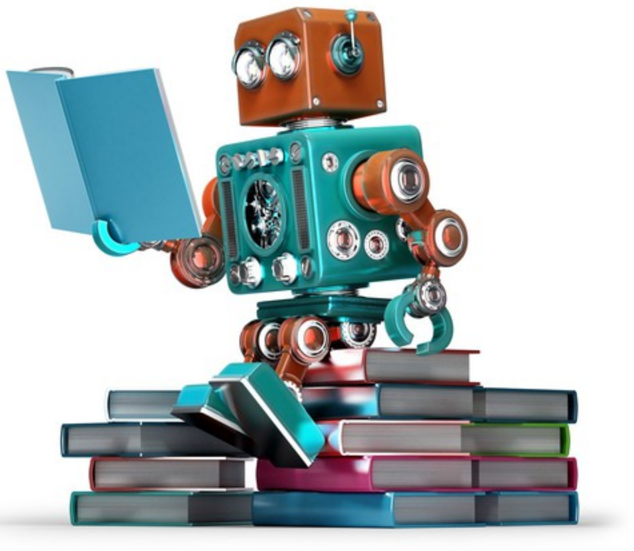
November 2020

Documents Semantic Analysis

- ctrl+f will never feel the same -



Author: Stéphane BAUDRY
Software Engineer M1.1

INTRODUCTION

This project has aimed at solving the issue of finding and graphically displaying relevant information within companies' suppliers and competitors' annual reports.
A regular word search would only target the specific terms while this application is able to understand the context of the search and return anything related to that context.
For example, a search for autonomous car does not only return paragraphs containing the combination of "autonomous" + "car" but also return anything related to "Artificial Intelligence" + "car"; since "autonomous" can appear in the context of "machine" and "A.I."

This report provides a comprehensive description of the project, the need, our answer to it and implementation, a description of the database and how it is used, as well as peeks of the actual application that has been developed consequently.

1. Problem definition:

The client needs to extract relevant information from suppliers and competitors public annual reports and have a visual representation of the importance of those informations throughout the past years. Furthermore, the user must be able to instantly access the page where an information is located. The format of the report might be PDF, Word ou HTML.

2. Problem structure

-The document needs to be loaded on the platform and processed, which means the application needs specific libraries to decompose and process text,
-The extracted data needs to be stored,
-Each document and data stored must have unique identifiers and metadata, such as year and language,
-The data must efficiently organised,
-The problem implies the use of an artificial intelligence algorithm, which includes a vectorizer and a classifier,
-the user must be able to interact with a platform and get a visual on the trend specific to their search, as well as a way to reach the desired documents and pages where information is located,
-The project must be exportable, independently of the Operating System,
-The project must be collaborative, hence must be in a cloud repository.

3. Assumptions

-A hierarchy structure could divide documents into corpus of document of similar content and then be subdivided by company, or 'categorised',
-It also implies the creation of a dataset for training, testing and validation,
-Assuming supervised learning, the data must be manually labelled and validated by experts,
-Assuming binary classification, the labelling must be done against something. Therefore, it requires some "topics" to be defined prior to the labelling stage; those topics could even be divided into sub-topics,
-By extension, you must be able to create and delete, topics and subtopics,
-The labelled dataset needs to be stored for reuse,
-To take into consideration human labelling errors and model classification errors, the mislabelled data could be manually corrected following a training,
-We assume a French as well as English user interface, switchable directly from a cascade button,
-The user can select the categories (companies) and years so as the trends displayed are within a specific timeframe,

-Each component of the web app, i.e. user interface, webservices, REST API and phpMyAdmin, could be placed in individual containers,
-Those containers could all built at once with docker compose.

4. The development process at a glance

i. Developing the database

- Define database paradigm needed: MySQL (relational) or document-oriented (NoSQL)
- Gather a pool of documents
- Create tables
- Handling requests - creating Views
- Process and store documents in database, including metadata
- Develop data quality checks

ii. Using the database

- Test database request and quality of data returned
- Get a visual representation of the database content at an instant t

iii. Project management

- Version control: create a repository
- Log difficulties. bugs, resolutions and sources used

iv. Building a classifier

- Create dataset for training the model
- Control datasets characteristics
- Hyperparameters research
- Train the model and get metrics
- Define possible optimizations

v. Developing the web app

- Frontend
    - Flask, templates, CSS
- Backend
    - Django, webservices, views - Object-relational Mapping (ORM)

5. Development – Agile

The project is divided into ‘sprints’ of 3 weeks, the backlog is refined in the middle of each sprint to see if the client’s need is correctly answered. Furthermore, daily meetings give a visual on the team advances as well as roadblocks. Weekly meeting also surveys the overall advancement.

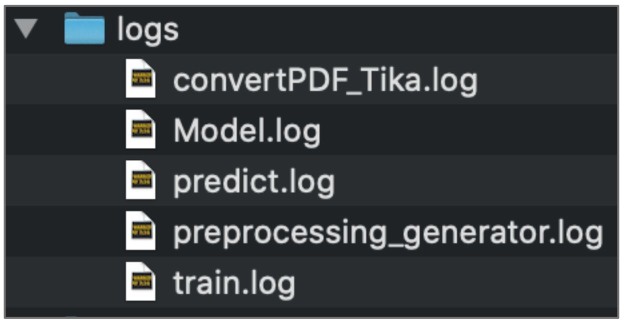
6. Project management

vi. Version control

The project’s repository is on **Gitlab** since it seems to have better features for **authentications** and also as the project scales, there might be a need for **CI/CD** (Continuous Integration/Continuous Delivery) in the future. We used a tool named **Tortoise**, which is a user interface for git commands, to interact with Gitlab. At the end of each sprint, different branches, that were used to work on some features, are merged and the final main branch is tested, and conflicts are resolved.

vii. The logs

Everytime a document is added, the programme log different steps such as the conversion PDF-XML-PICKLE, the functions called to make requests, different steps of the normalization process, the training and predict phase, as seen in Figure 5:



```
logs
    convertPDF_Tika.log
    Model.log
    predict.log
    preprocessing_generator.log
    train.log
```

Figure 5. Logs

Bugs resolution: main sources are **Stakoverflow** and **Django official website**

7. The database

viii. Relational vs Document-Oriented database

In a nutshell, **relational databases** are very useful for situations where you might have a lot of data that intersects with each other, though it takes a **fair bit of planning** as each field must have the **datatype specified**, and in the case of strings you also need to declare the maximum **character limit**.
Document databases don't require the careful planning of a SQL database but are more limited in their use. They instead store data in collections consisting of documents (something like a JSON). This offers a lot **more flexibility** as you don't have to meticulously plan out your data into tables, but this comes at the **cost of speed and size**. Since you're just **storing full-on objects**, you'll have to **pull them out** of the database and into your application instead of just querying what information you're looking for in particular like in a SQL database.

ix. MySQL

Speed. MySQL is fast. it might even be the fastest database system you can get. (see http://dev.mysql.com/tech-resources/benchmarks/, a performance-comparison page on the MySQL AB Web site),

Ease of use. MySQL is a high-performance but relatively simple database system and is much less complex to set up and administer than larger systems,

Query language support. MySQL understands SQL (Structured Query Language),

Capability. The MySQL server is multi-threaded, so many clients can connect to it at the same time. Each client can use multiple databases simultaneously,

Connectivity and security. MySQL is fully networked, and databases can be accessed from anywhere on the Internet, so you can share your data with anyone, anywhere. Also, MySQL supports encrypted connections using the Secure Sockets Layer (SSL) protocol,

Portability. MySQL runs on many varieties of Unix, as well as on other non-Unix systems,
Small size. MySQL has a modest distribution size, especially compared to the huge disk space footprint of certain commercial database systems,

Availability and cost. MySQL is an Open Source project with dual licensing. First, it is available under the terms of the GNU General Public License (GPL). This means that MySQL is available without cost for most in-house uses. Second, for organizations that prefer or require formal arrangements or that do not want to be bound by the conditions of the GPL, commercial licenses are available,

Open distribution and source code. MySQL is easy to obtain; just use your Web browser.

x. phpMyAdmin

Having a visual on the database and being able to interact with it without the any prompts to enhance understanding is a gain of speed and productivity.
phpMyAdmin is a free software tool written in PHP, intended to handle the administration of MySQL over the Web. phpMyAdmin supports a wide range of operations on MySQL. Frequently used operations (managing databases, tables, columns, relations, indexes, users, permissions, etc) can be performed via the user interface, while you still have the ability to directly execute any SQL statement.

8. The data

Public annual reports of a few different companies from the water and energy industry are gathered. It ranges from 2015 to 2019.
Each document is in PDF format an unprotected.
On average, the length of documents is 100 pages with around 30,000 words each

9. Creating the database

xi.Django

This project requires the development phase to show both efficiency and most importantly speed, at least at its early stages; project, which could still be scaled relatively easily.

Django's models provide an Object-relational Mapping (ORM) to the underlying database. ORM is a powerful programming technique that makes working with data and relational databases much easier.
You just need to create both your models and serializers, then via your browser Django takes care of the requests through functions in a 'views' script, as seen in Figure 1:

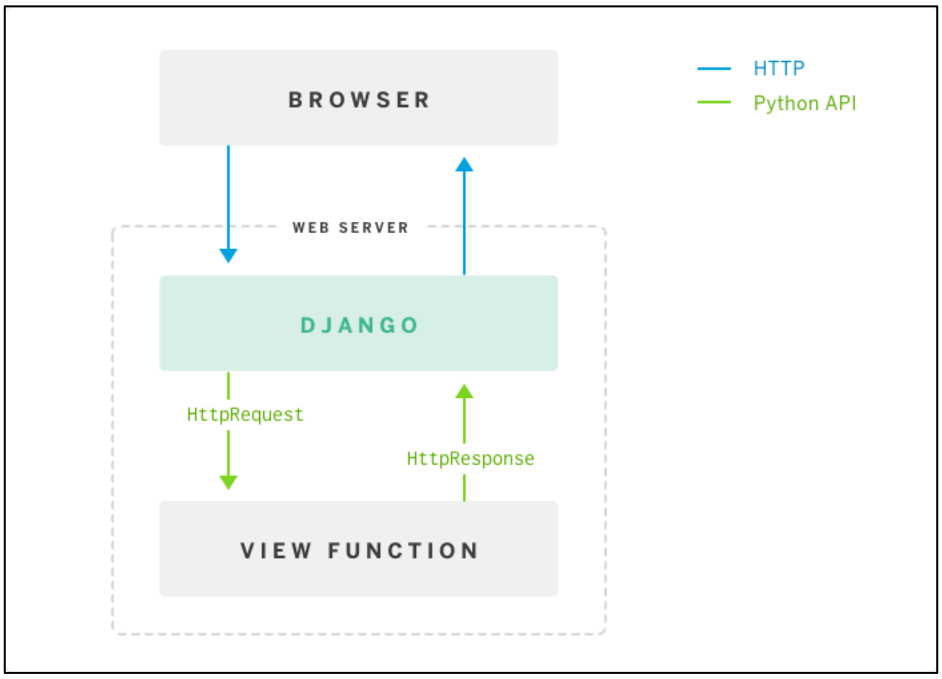


Figure 1. Django requests architecture

xii.Type Storage Requirements

(cf. MySQL 8.0 Reference Manual: 11.7 Data Type Storage Requirements)
Also, an description of Bits and Bytes by Stanford: https://web.stanford.edu/class/cs101/bits-bytes.html

- **Numeric**: such as IDs of documents, page number, sentence number, etc.
  So we know 1 byte is a group of 8 bits that can make 256 patterns (0 to 255).
  IDs of documents should not go up too crazy and except for ID_Sentence, other IDs are repetitive. Therefore, we use '**bigint**' (8 bytes) for ID_Sentence and simple '**int**' (4 bytes) for the rest.

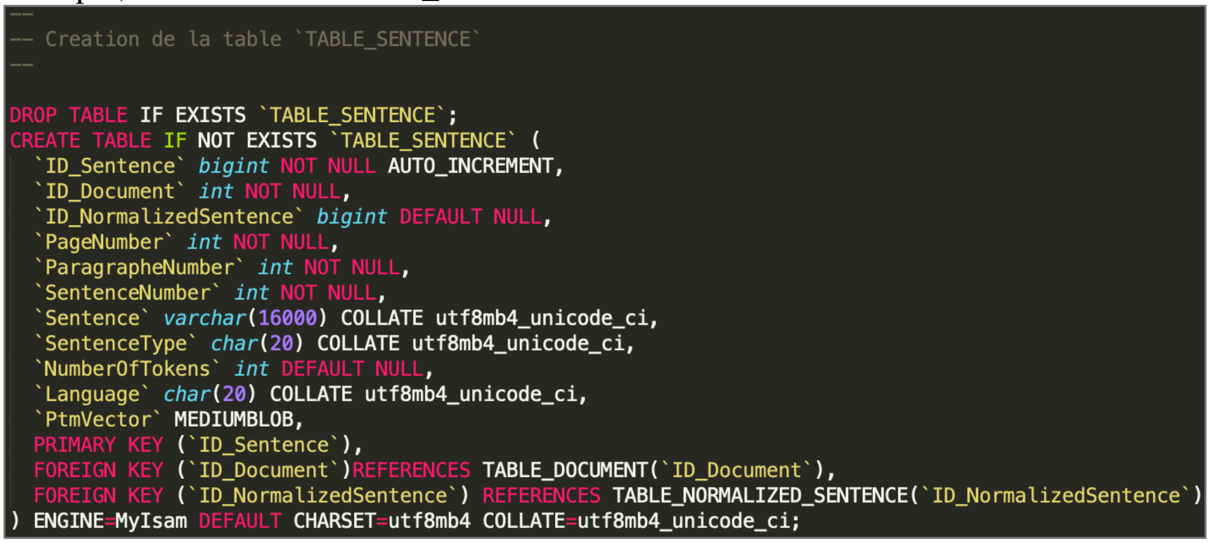
- **Date and time**: such as date of creation and modification of a table.
  The wanted format would be something like '2020-10-26 11:34:53'. Interesting fact from an external media Sharpcorner.com: "Supported range for DATETIME is '1000-01-01 00:00:00' to '9999-12-31 23:59:59' while for TIMESTAMP, it is '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC". Therefore, for now we decided to go with 8 bytes '**datetime**'.

- **Variable-length types**: such as raw sentence, vector array (768 dims.), language, etc.
  An analysis if the extracted sentences shows an average length of 25 words, although some cases can be much longer. Hence, '**varchar'** is chosen to store sentences. Also, vectors of size 768 needs to be stored, therefore we decided to go

with ‘**MEDIUMBLOB**’ which is L + 3 bytes, where L < 2^{24}, where L represents the actual length in bytes of a given (string) value. It also seemed like we had to specify ‘**utf8mb4_unicode_ci’** to store characters different from English, for example in case of extracting sentences from a French document.

xiii.Creating tables

We create SQL commands in a .sql that we then import from phpMyAdmin directly. For example, for the table “TABLE_SENTENCE”:



```
--
-- Creation de la table `TABLE_SENTENCE`
--

DROP TABLE IF EXISTS `TABLE_SENTENCE`;
CREATE TABLE IF NOT EXISTS `TABLE_SENTENCE` (
  `ID_Sentence` bigint NOT NULL AUTO_INCREMENT,
  `ID_Document` int NOT NULL,
  `ID_NormalizedSentence` bigint DEFAULT NULL,
  `PageNumber` int NOT NULL,
  `ParagrapheNumber` int NOT NULL,
  `SentenceNumber` int NOT NULL,
  `Sentence` varchar(16000) COLLATE utf8mb4_unicode_ci,
  `SentenceType` char(20) COLLATE utf8mb4_unicode_ci,
  `NumberOfTokens` int DEFAULT NULL,
  `Language` char(20) COLLATE utf8mb4_unicode_ci,
  `PtmVector` MEDIUMBLOB,
  PRIMARY KEY (`ID_Sentence`),
  FOREIGN KEY (`ID_Document`)REFERENCES TABLE_DOCUMENT(`ID_Document`),
  FOREIGN KEY (`ID_NormalizedSentence`) REFERENCES TABLE_NORMALIZED_SENTENCE(`ID_NormalizedSentence`)
) ENGINE=MyIsam DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

Figure 2. SQL to create TABLE_SENTENCE

Handling requests - creating Views
What are Views and why using them? As per Django official website:

“A **view function,** or **view** for short, is a Python function that takes a **Web request** and returns a **Web response**. This response can be the HTML contents of a Web page, or a redirect, or a 404 error, or an XML document, or an image . . . or anything, really. The view itself contains whatever arbitrary logic is necessary to return that response.”

also, from the website The Django Book:

“Django’s views are the information brokers of a Django application. A view sources data from your database (or an external data source or service) and delivers it to a template. For a **web application**, the view delivers **webpage content and templates;** for a **RESTful API** this content could be formatted **JSON data**.

The view decides what data gets delivered to the template—either by acting on input from the user or in response to other business logic and internal processes. Each Django view performs a specific function and has an associated template.”

In other words…it makes our lives easier.
-Views provide abstraction over tables. You can add/remove fields easily in a view without modifying your underlying schema
-Views can model complex joins easily.

Adding documents to the database
Initially, documents were added through **Django REST API** where we specified the name of the folder in a json command: {“corpus_name”:”corpus_pdf”}. Today, each document can be added through a webpage where the user specifies a corpus name, a category, the year of the document, the language and can browse his machine to select the document.

xiv.Pre-processing

Once the document is submitted, it enters a data extraction phase. The document is sliced in pages, then paragraphs, then phrases. To do this, we use Apache Tika. Apache Tika is a content detection and analysis framework, written in Java, it detects and extracts metadata and text from over a thousand different file types. The metadata collected (page, paragraph, sentence) is important as it will be used to reach out the correct page of the information needed at the end.

xv.Processing

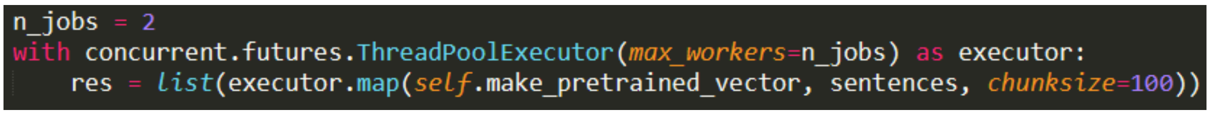
This step “cleans” the text and reduce its size through **lemmatization** or **stemming** depending on your parameters in the configuration file. For English, speed and performance seems to be better with the **NLTK** library, while the **SPACY** library seems to be more efficient with other languages (lemmatization and **stopwords**).
Once cleaned, each sentence receives its vectorial representation. In term of ease of use and performance, **SentenceTransformers** was the best bet. According to the official website:

“SentenceTransformers is a Python framework for state-of-the-art sentence and text embeddings. The initial work is described in our paper Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks.”

This library is particularly useful as our project works with documents in many languages and when you use SentenceTransformers, the main model is pretrained for English text, and the multilingual model, i.e. “**distiluse-base-multilingual-cased**”, is effortlessly applied to many other languages.
Once each sentence receives a 768 dimensions vectorial representation. It is saved in the database as a “BLOB”.

A **BLOB or Binary Large Objects** is “designed for efficient storage of large objects. Without BLOB support, large objects must be broken up into smaller pieces, and then reassembled and/or disassembled every time the record is read or updated.”

To gain speed, the project also uses the **Threading** technique to parallelize the vectorization task:



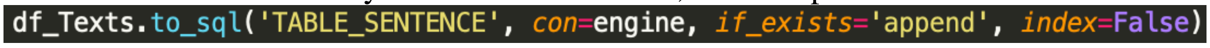
```
n_jobs = 2
with concurrent.futures.ThreadPoolExecutor(max_workers=n_jobs) as executor:
    res = list(executor.map(self.make_pretrained_vector, sentences, chunksize=100))
```

Figure 3. Parallelising the vectorisation step

In fine, several **Pandas** dataframe are created: df_Texts, df_Norm, df_meta
df_Texts contains data relative to each sentence such as document, Page Number, Id_sentence, vector, etc.

While, df_Norm is used to store lemmatized sentences, and df_meta to store metadata such as supplier name, id_supplier, year, type of documents, etc.

These dataframes are finally added to the database, for example:



```
df_Texts.to_sql('TABLE_SENTENCE', con=engine, if_exists='append', index=False)
```

Figure 4. Adding the dataframe to the database

xvi.The distribution of data

Figure 6 and Figure 7 below, describes how much a category (a company) contributes to the 35,000 sentences in the database:

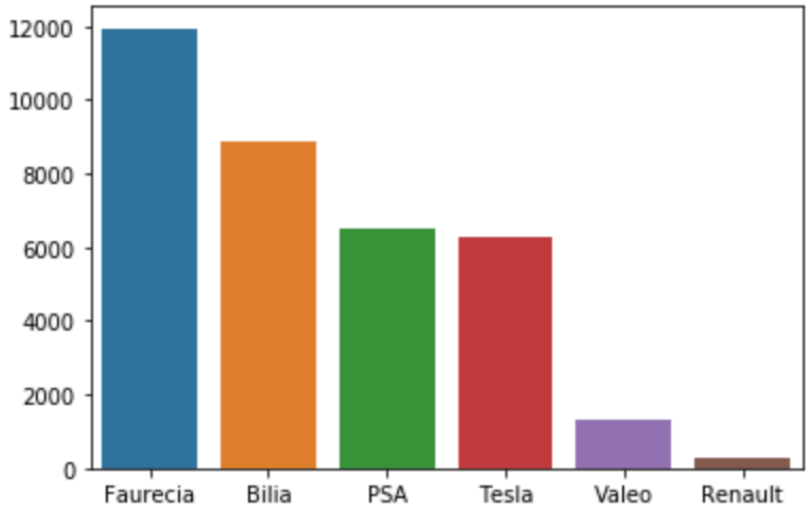
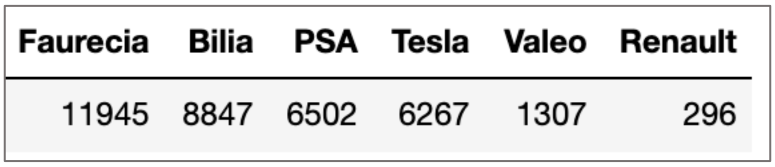


Figure 6. Distribution of sentences extracted per category

As seen on the table below, Faurecia has very long reports where 11,945 sentences have been extracted out of 3 documents. On the other hand, Tesla reports are much shorter with 6,267 sentences extracted out of 5 documents. The unbalance does not matter here, because we can assume that most manufacturer refer the same way on a piece of technology.



Faurecia	Bilia	PSA	Tesla	Valeo	Renault
11945	8847	6502	6267	1307	296

Figure 7. Count of sentences extracted per category

xvii.Optimising the database

After inspection of the database, some sentences were repeating, as seen in Figure 8

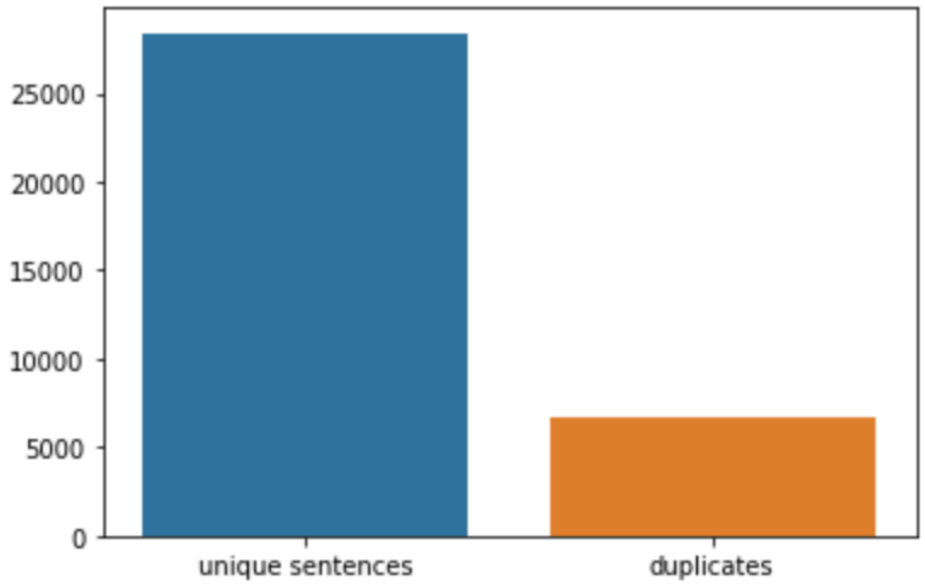


Figure 8. Number of duplicates in the database

As seen on the figure above, over 6,000 sentences were duplicated, which was due to some inconsistency with the **.jar** file of **Apache Tika**, which has troubles when some images are present at the same with the text.

To fix the issue, we implemented a couple of checks for duplicates: while the PDF is being converted to text, processed and serialized and a second check once the data has been uploaded to the database:

```
df_sent['Sentence'].duplicated().value_counts()
```

If duplicates are counted, they are dropped and the clean dataframe is sent to the database:

```
df_sent['Sentence'].duplicated().value_counts()
df_sent.to_sql('TABLE_SENTENCE', con=engine, if_exists='replace', index=False)
```

Same thing, for null values, sentences are checked before and after entering the database:

```
df_sent['Sentence'].isna().value_counts()
```

10.Creating the dataset

Introduction
Before a document is added in the database, the user specifies a corpus name (e.g. “Autonomous Car” or “Water and Electricity”) that represent a cluster to which documents can be subdivided into “category”, namely, the name of the company (e.g. “PSA”, “SUEZ”, etc.)
Once a document is added, the user needs to define a “Topic” and “subtopics”, for example, having the French corpus “Eau” and category “Suez”, a topic of interest could

be “Pollution” and its subtopics could be “traitement des eaux”, “contamination”, “pathogène” for example.

xviii.Selecting phrases close to the topic

As soon as a subtopic is created, a TF-IDF is applied to all sentences of a particular Corpus, and the ones selected are added to the phrases to be labelled later on.
A new feature also, that can automatically generate subtopics for the user, takes a topic description, vectorize it with SentenceTransformer, look at the distance between that vector and all vectors of the Corpus. A TF-IDF is applied to the closest ones and the <top-n> <n-gram> are returned to the user with the number of sentences to be labelled for each subtopic.

xix.Labelling

The classification is binary, with “YES” if a sentence matches the topic or “NO” if it isn’t. We start with 50-100 phrases labelled for each topic.

xx.Distribution of labelled data

The barchart in Figure 9 represents the percent of each label in the database:

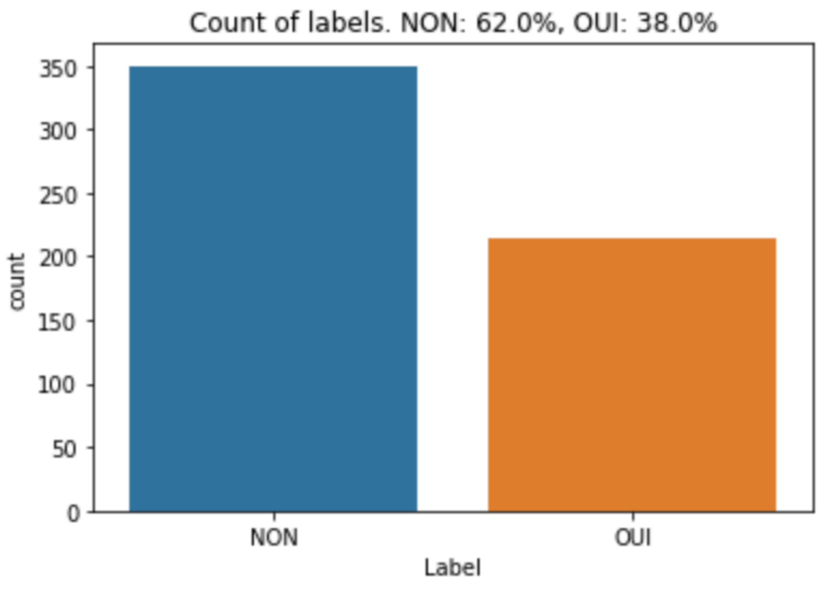


Figure 9. Distribution of labels in the database

Hence, we can see a majority of NOs as an annual report talks about much more than simply investments in A.I. technology, therefore this distribution seems fair.

The split
After several experimentations, a split 70/20 seems to bring the best results, which gathers the following:

```
x_train, y_train: 406, 406
x_test, y_test: 113, 113
```

11.Building the classifier

First, the labelled data is split between train and test: 85% train, 15% test.
Then, we use a simple but efficient classifier: **Random Forest**.

xxi.The model

Why Random Forest over a **Neural Network** for example. A Random Forest is less **computationally expensive** and does not require a **GPU** to finish training. Neural Networks will require much more data than an everyday person might have on hand to actually be effective. Also, the neural network will simply decimate the **interpretability** of your features to the point where it becomes meaningless for the sake of performance.

xxii.Hyperparameters Grid-Search

```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier(random_state=40)

param_grid = {
    'n_estimators': [50, 100, 200, 300, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8,10],
    'criterion' :['gini', 'entropy']
}

from sklearn.model_selection import GridSearchCV
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x_train, y_train)
```

Which results in:
max_features = log2
n_estimators = 120
max_depth = 6
Criterion = entropy

```
rfc1=RandomForestClassifier(random_state=41, max_features='log2',
                            n_estimators= 120, max_depth=6,
                            criterion='entropy')
```

xxiii.Training the model

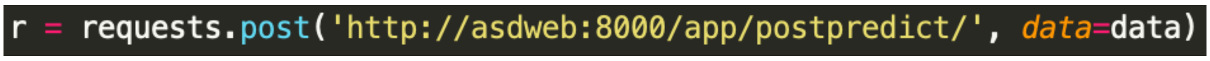
We can send a request to launch the training:

```
r = requests.post('http://asdweb:8000/app/posttrain/', data=data)
```

We train the model with the aforementioned parameters.

```
model = RandomForestClassifier(n_estimators=int(clf_params['n_estimators']),
                               max_features=clf_params['max_features'],
                               random_state=42)
model.fit(pipe_vect.transform(x_train), y_train)
```

The prediction follows:

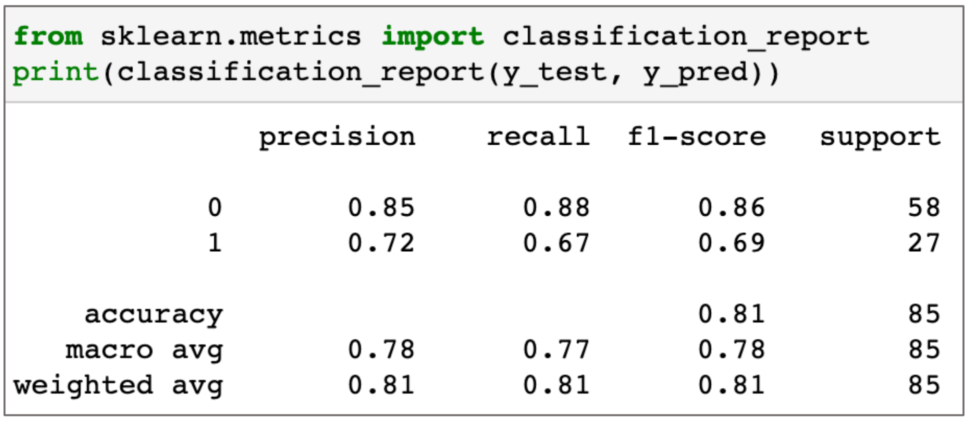


```
r = requests.post('http://asdweb:8000/app/postpredict/', data=data)
```

xxiv.Metrics

11.11.1.Classification report & confusion matrix

Comparing the prediction to the true value, the classification report (Figure 10) and confusion matrix (Figure 11) is as follow:



```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.88      0.86        58
           1       0.72      0.67      0.69        27

    accuracy                           0.81        85
   macro avg       0.78      0.77      0.78        85
weighted avg       0.81      0.81      0.81        85
```

Figure 10. Classification report

Precision (or positive predictive value) tells us that the model only mislabelled 'YES' 6 elements, bringing the score to 85%.

Recall (or sensitivity) tells us that the model correctly found 88% of all existing 'YES'.

F1 is better used to compare different models, not the global accuracy of one model. Nevertheless, as a weighted harmonic mean of precision and recall, we get a score of 86%.

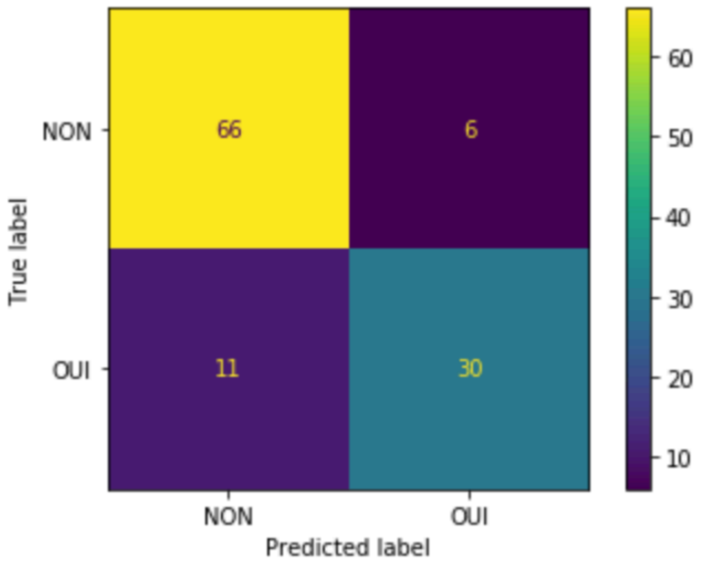


Figure 11. Confusion matrix

The **accuracy** tells us that out of all the prediction made, 81% are correct.

Finally, **specificity** can also be calculated to give us the amount of True Negative that the model actually found out of all existing negatives; i.e. TN/(TN+FP): 92%

11.11.2.ROC Curve

As seen in Figure 12, our classifier's Receiver Operator Characteristic (ROC) curve shows a decent trade-off between sensitivity (or TPR) and specificity (1 – FPR) and Area Under the Curve (AUC) of about 89%.

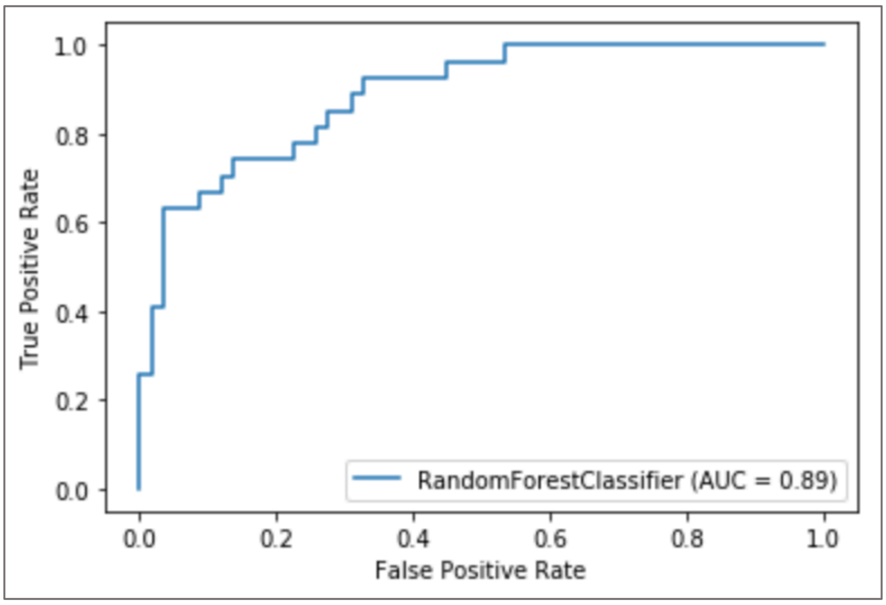


Figure 12. ROC Curve

xxv.Possible optimizations

-How sentences are extracted and normalized needs more work as some sentences in the databases are truncated, mostly due to the fact that in the PDF, those phrases arrive at the end of a page and restart on the next one;
-More sentences need to be labelled;
-Cosine distances between BERT vectors would be better than TF-IDF;
-Gradient Boosting could replace Random Forest

12.Development of the web app

xxvi.Flask+Django

The application was initially built solely with Flask and the data were saved in CSVs. Now, The User Interface still uses Flask, and the backend is handled with Django, as seen in Figure 13.

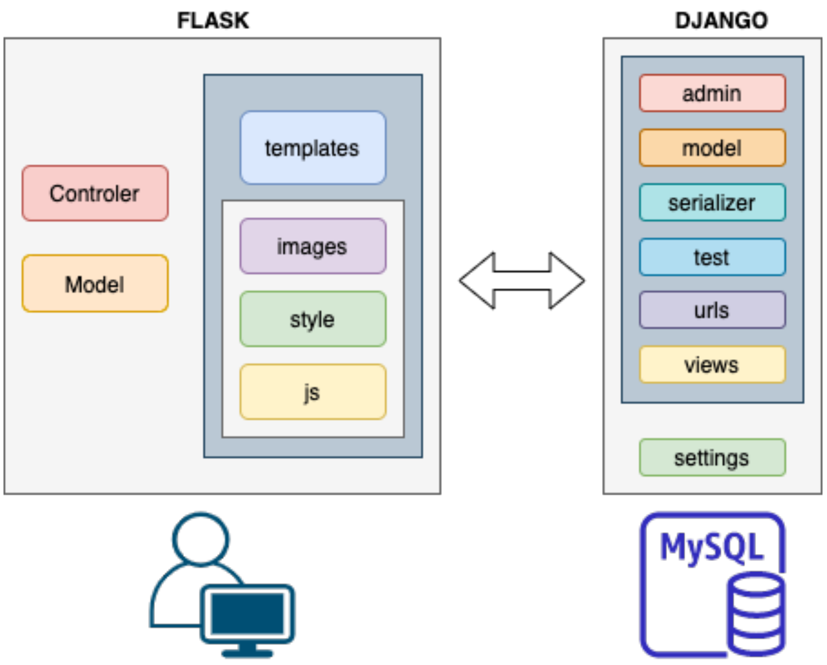
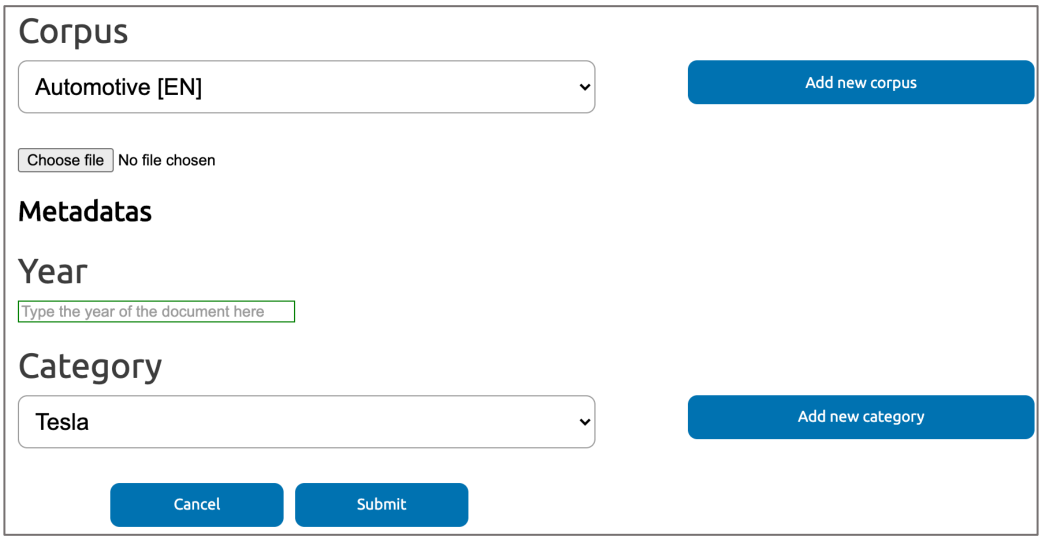


Figure 13. The application's model

xxvii.User interaction with the app

If the user has never added a document, Figure 14 is the first window to appear. The user is prompt to add a corpus, a category, the year of the report and browse the document on their machine.



Corpus

Automotive [EN]

Choose file No file chosen

Metadatas

Year

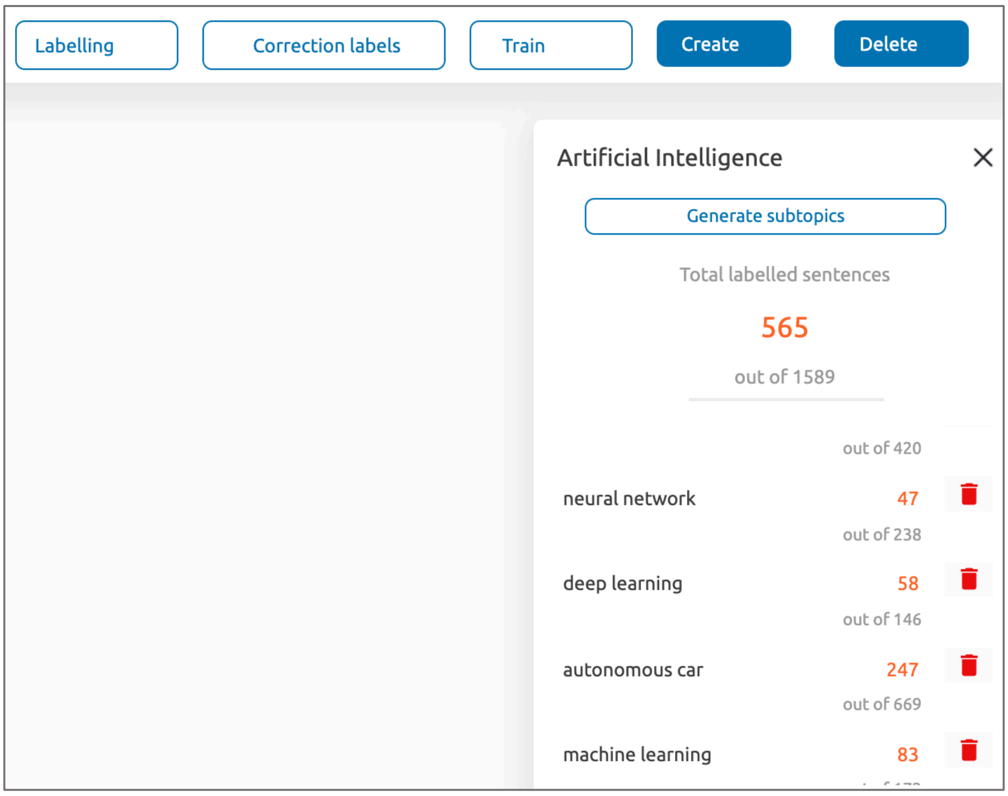
Type the year of the document here

Category

Tesla

Figure 14. Add a document

Once, the document is processed and added the database, it is time to create a Topic as well as subtopics, see Figure 15. The platform will automatically calculate the volume of sentences that seems related to the topic.



Total labelled sentences

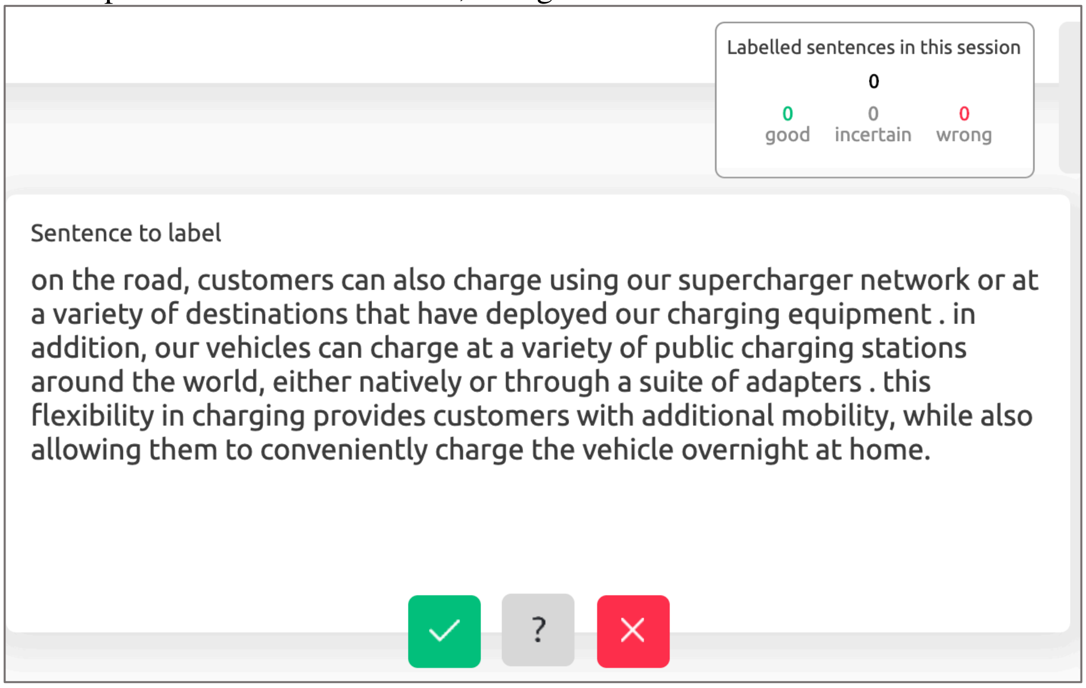
565

out of 1589

	out of 420	
neural network	47	🗑
	out of 238	
deep learning	58	🗑
	out of 146	
autonomous car	247	🗑
	out of 669	
machine learning	83	🗑

Figure 15. Add a topic and subtopics

The next step is to label our sentences., as Figure 16 shows:


Labelled sentences in this session

0

0	0	0
good	incertain	wrong

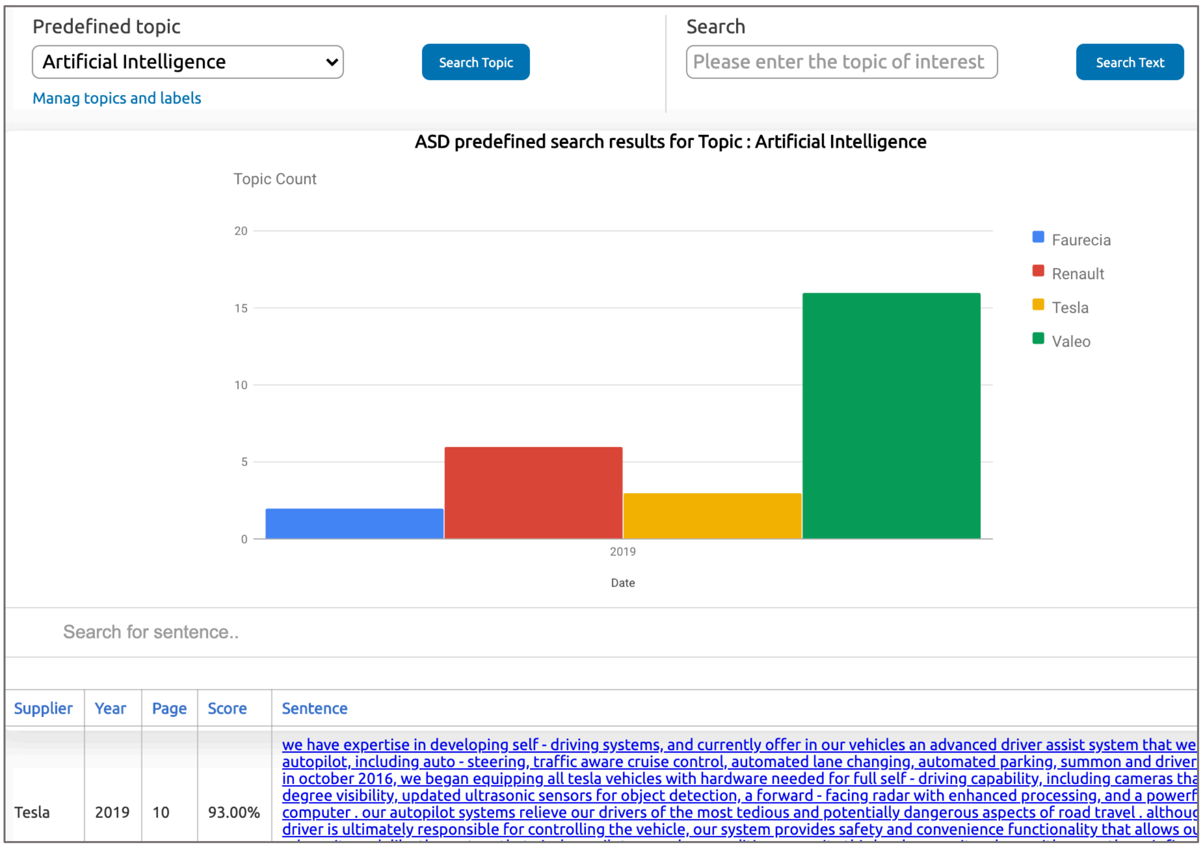
Sentence to label

on the road, customers can also charge using our supercharger network or at a variety of destinations that have deployed our charging equipment . in addition, our vehicles can charge at a variety of public charging stations around the world, either natively or through a suite of adapters . this flexibility in charging provides customers with additional mobility, while also allowing them to conveniently charge the vehicle overnight at home.

Figure 16. Labelling page

Next, the user launches the training phase by pressing the Train button. Once done, the user can return to the main page. It can add new documents and make predictions based on the topic created and trained previously.

Finally, Figure 17, the prediction generates a graphic that can show the importance of the topic for companies and it also provide direct links to the document's page where the information is located.



Supplier	Year	Page	Score	Sentence
Tesla	2019	10	93.00%	we have expertise in developing self - driving systems, and currently offer in our vehicles an advanced driver assist system that we autopilot, including auto - steering, traffic aware cruise control, automated lane changing, automated parking, summon and driver in october 2016, we began equipping all tesla vehicles with hardware needed for full self - driving capability, including cameras tha degree visibility, updated ultrasonic sensors for object detection, a forward - facing radar with enhanced processing, and a powerf computer . our autopilot systems relieve our drivers of the most tedious and potentially dangerous aspects of road travel . althouc driver is ultimately responsible for controlling the vehicle, our system provides safety and convenience functionality that allows ou

Figure 17. Prediction page

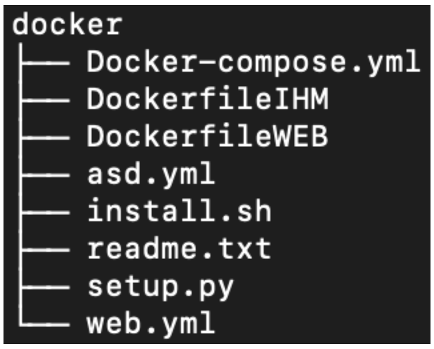
13.Containerisation – Docker Compose

What is docker-compose? As per Docker documentation:

"Compose is a tool for defining and running **multi-container** Docker applications. With Compose, you use a **YAML** file to configure your application's services. Then, with a single command, you create and start all the services from your configuration.

Compose works in all environments: **production, staging, development, testing**, as well as **CI workflows**."

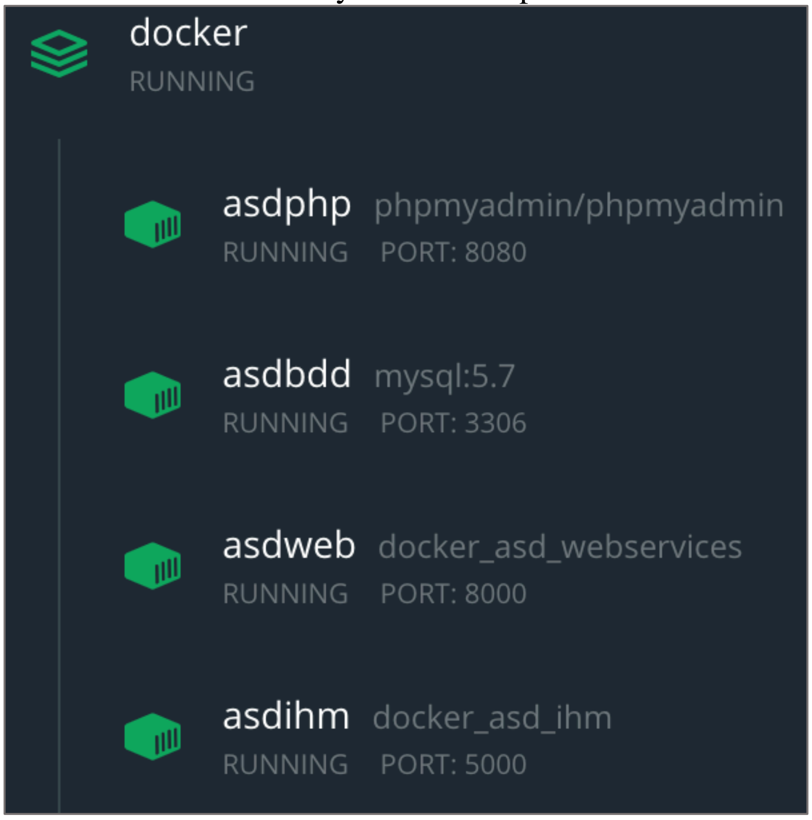
Setting up the creation of all containers is a bit of work but once done, the project is organised, you can start the whole project with one command, i.e. docker-compose up, you do not need to run all the containers, you start some and stop others, hence easing on the memory. Figure 18 shows how the docker folder is organised.



```
docker
├── Docker-compose.yml
├── DockerfileIHM
├── DockerfileWEB
├── asd.yml
├── install.sh
├── readme.txt
├── setup.py
└── web.yml
```

Figure 18. Docker folder

Figure 19 shows the container created by docker-compose.



```
docker
RUNNING

asdphp  phpmyadmin/phpmyadmin
RUNNING   PORT: 8080

asdbdd  mysql:5.7
RUNNING   PORT: 3306

asdweb  docker_asd_webservices
RUNNING   PORT: 8000

asdihm  docker_asd_ihm
RUNNING   PORT: 5000
```

Figure 19. The 4 containers created from docker-compose