Description of Use Cases

SG Artificial Intelligence Study

Editor/Author Name	Initials	Organisation / Email
[redacted]	[redacted]	DIGIT.D.1
[redacted]	[redacted]	DIGIT.D.1
[redacted]	[redacted]	DIGIT.D.1
[redacted]	[redacted]	DIGIT.D.1

Contents

This document outlines DIGIT.D.1's proposal for four (4) Proof of Concepts PoC (will also be noted as "use cases") including their description, the steps undertaken and their outcome.

PoC 1: Personal data recognition in text

Using Personal Data Recognition the user will be able to identify personal information in a text. The interface will identify, where possible, the person and its grade. With this tool, the user will be able to execute a predefined action to amend a text and generate a new version of the document.

Why use A.I?

Artificial Intelligence's biggest advantage is its ability to learn and generalize rules from data. The learned rules are then applied on new unseen data for different purposes.

When working with documents the main challenges usually are:

- Large amount of data to be processed
- Repetitive tasks
- Different formats and Languages
- Duplicates of contents
- Different "styles" and context of content
- Quality
- Time to deliver

A.I has the ability to adapt to any particular "style" and context to extract rules.
These rules can be used to:

- Identify points of interest in the documents
- Identify similar parts of documents
- Identify similar documents
- Enforce 'style'
- Verify Quality
- Create or translate documents
- Provide a continuous and improving quality of delivery

By combining A.I with current state of the art computational power:

- Large amount of documents can be processed extremely fast
- Data can be processed 24/7 with equal quality of delivery

By combining A.I with human verification and interaction, systems that merge speed, accuracy, continuous quality with human abilities can be built. Moreover providing feedbacks to the A.I will allow the system to continuously evolve. By using this approach a so-called *human-in-the-loop systems* is built.

Step 1: Creation of a dataset

First a dataset that identifies the start and end positions of personal data in text is annotated (labelled). Then the infrastructure and functionalities to securely move the data to the place of computation is prepared. Finally, the data is split into two subsets. A train subsets and a validation subsets.

Usually personal data refers to:

- Name
- Surname
- E-mail
- Postal address
- Phone number
- Fax-Number
- Job title
- (optional) Handwritten initials
- (optional) Handwritten signatures
- (optional) Photos

Step 2: Training and optimisation of the algorithm

During this stage, the dataset will be used to train an algorithm to identify the parts of the text labelled as personal data. Once a sufficient accuracy is achieved, the weights of the algorithm will be froze into a model. This model will then be used for the integration of the algorithm into the further proposed functionalities.

Step 3: Association of classified personal data to Who is Who

During this stage, the functionalities to link the data classified by the model to *Who is Who* will be created in the effort to provide the possibility to identify the administrative position of the person mentioned in the text.

Step 4: User interface

A simple interface allowing the analysis of one or multiple files will be developed. For every file, the user will be able to visualize the personal data identified by the algorithm. The visualisation will present the value, the class predicted by the algorithm and the information retrieved from *Who is Who*. Moreover, the user will be able to select the entries he wishes to modify and execute the desired modification(s) on all of them.

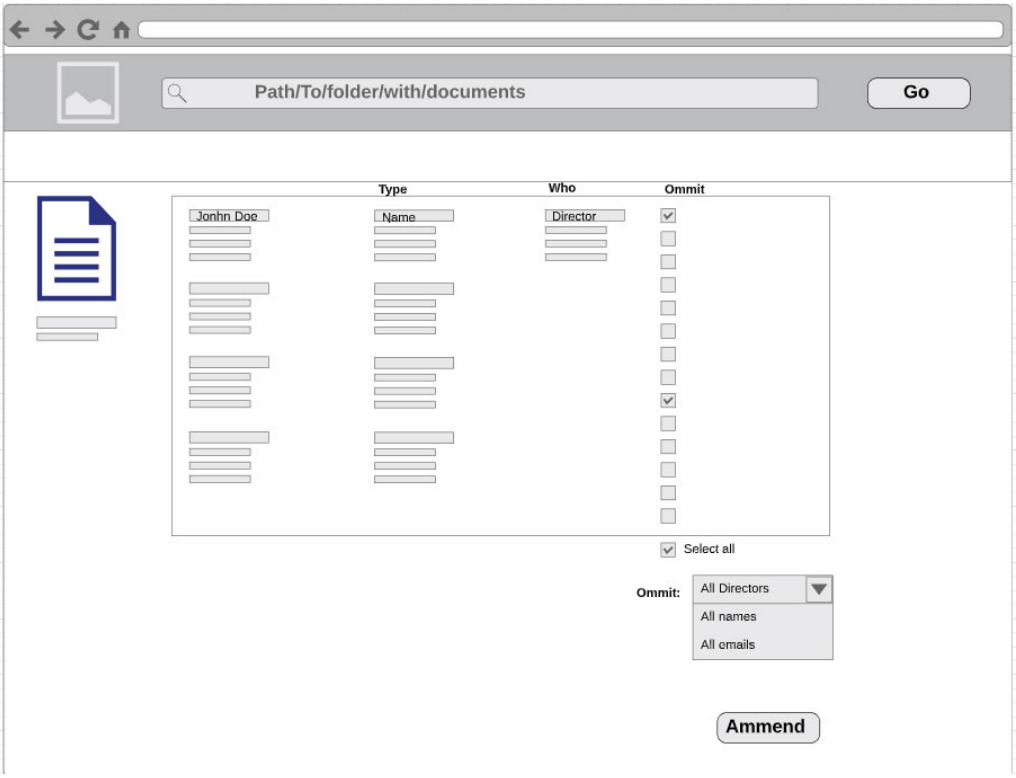
User scenario

Francine is in charge of removing personal information from documents that are sent out by her unit. Her task is to read all the documents that relate to specific request and indicate where in the text personal information is mentioned. Francine will then go to an internal application to find out what the persons role is. Given the specificities of the request she needs to make some or all types of personal data unreadable in respect of the persons role.

Instead of doing this manually, Francine can drop the files in a folder and indicate the path to follow to

find the folder.

The documents will be analysed using artificial intelligence. Francine can open an interface and visualise all the personal data found in each document.
Next to the personal data she can also see information retrieved from internal service indicating the job title of the person. She can select one or more entries and select a modification function that will be executed on the text and generate a modified version of the document.



Technology

To achieve the above-mentioned functionalities the text needs to be fed as an input to a deep learning algorithm. This algorithm is then being asked to 'predict' the classification of every word or a sequence of words in the text.

When working with words and letters, these words and letters needs to be fed into the algorithm in order to transform them into arrays of numbers. The same method is applied with images. In order to achieve this, a vocabulary where every word is given a unique key is created.

Let's say we have a vocabulary of 3 words. Each of them has a unique number.

Rome → 1, Paris → 2, Italy → 3

At this point a *one-hot encoding* can be created to represent these words. *One hot encoding* represents a vector (an array) that has the length of the vocabulary's size. Meaning if there is a four words vocabulary each words will have an array of size four [0,0,0,0]. The word represented is written by using a '1' at the index of the word in the vocabulary.
For example:

Rome = [1,0,0,0,0]
I = [0,1,0,0,0]
Went = [0,0,1,0,0]
To = [0,0,0,1,0]
Italy = [0,0,0,0,1]

At this point a sentence (i.e. a sequence of words) can be represented by using vectors (arrays). The sentence: 'I went to Rome' can be represented as a sequence of *one hot encodings* as follow:

[[0,1,0,0,0] , [0,0,1,0,0] , [0,0,0,1,0] , [1,0,0,0,0]]

This representation is then fed into an algorithm. However it is not an ideal representation. The vectors (arrays) contain mostly zeroes (sparse) meaning that a lot of data will be needed in order to train an algorithm.

Given two sentences 'I went to Rome' and 'I went to Italy'. The difference between them does not represent the relationship between Rome and Italy. The indices at which the words Rome [1,0,0,0,0] and Italy [0,0,0,0,1] are saved in the vocabulary are random and do not represent any relationship. To solve this issue an approach called word2vec is used. In this approach the word as well at its context are looked at. One technique to generate word vectors (embedding) is Skip-gram.

Skip-gram:
Skip-gram looks at the context in which a word appears by looking at the words surrounding this word.
For example given the following sentence:

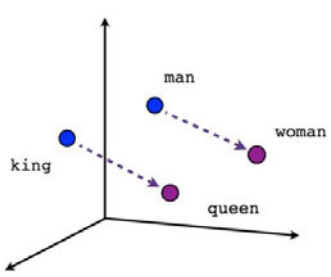
'I went to Rome'

The following pairs will be produced:

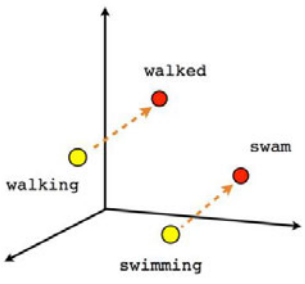
([I,to], went) , ([went,Rome],to)

These sequences can now be fed into a simple algorithm and trained using randomly generated negative samples (invalid sequences) and real sequences. The output of this algorithm is a dense vector (multi-dimensional array) called embedding. This is called semi-supervised learning.

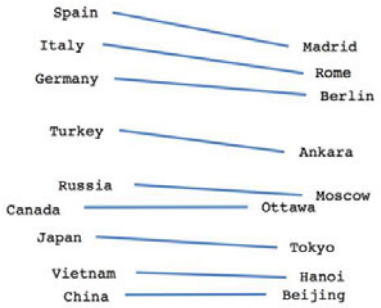
These embeddings are able to capture the relationships between words and when computing the distance (difference) between them the relationship is represented.



Male-Female



Verb tense

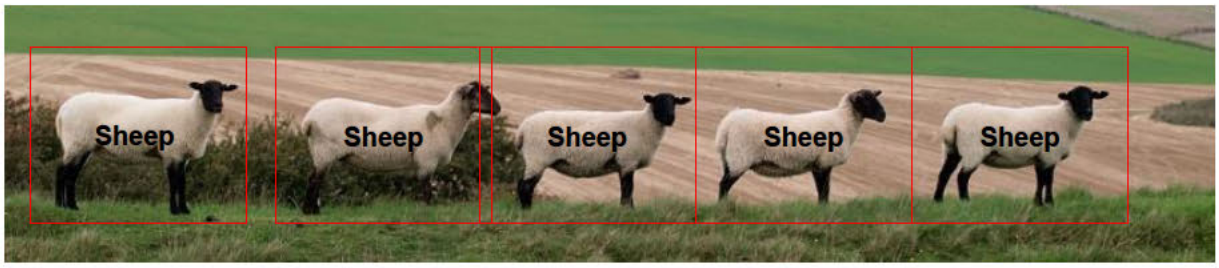


Country-Capital

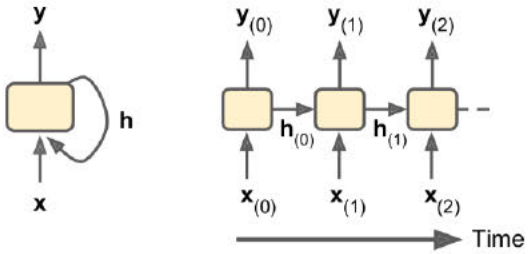
Skip-gram is not the only approach that can provide us with these kind of embeddings (context insights). Other possible approaches such as Glove and Bert are possible.

In our case we are looking for not only one word but a sequence of words. Most algorithms do not have a memory and are not able to take previous inputs into account to generate a prediction.

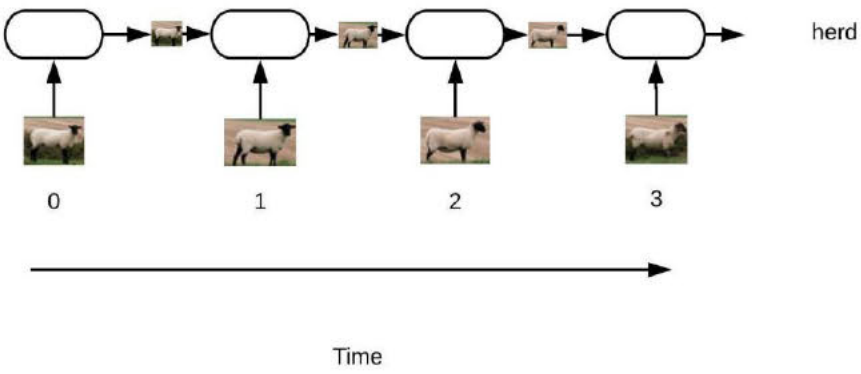
For example, an image classification algorithm can classify ten sheep, one after the other but it will not classify them as a herd of sheep.



An algorithm that can consider sequences and understand that seeing five sheep in a sequence means that it is looking at a herd is needed.
These algorithms are called recurrent neural networks RNN. What distinguishes them from the rest is that they take two inputs one of them being the previous output.



Using the sheep example:

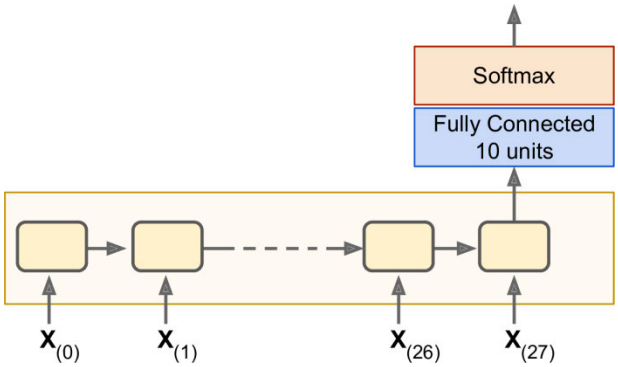


By using RNN classifications on sequences of inputs are made.

To achieve that we need to feed in the sequences into an RNN take in the outputs of all the sequences and pass them into a fully connected layer the output of which is used to make the classification.

A fully connected layer is a layer of neurons where all the neurons are connected to each other. To make the classification the Softmax function used. Softmax function provides a probability distribution over the

number of classes.



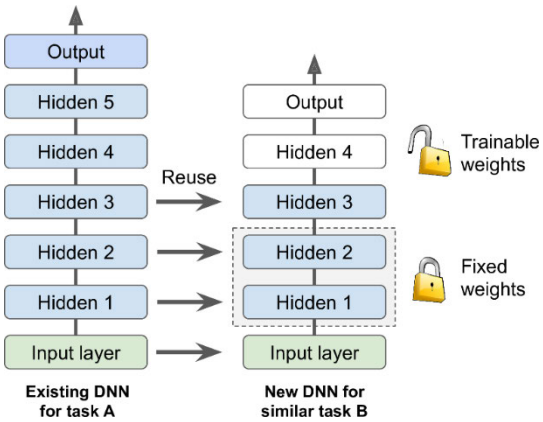
Using this approach a sequence of words is fed into an algorithm and teach it to extract rules to be able to classify the sequence.

Currently a number of open source NLP frameworks that provide this architecture exist as well as models that are already trained on large corpus of words. We are able to leverage on them by using a technique called transfer learning.

In deep learning the possibility to re-use a part of trained algorithm and to build on top of it is possible. It turns out that during training the lower layers of the algorithm will acquire the ability to distinguish basic rules and the higher layers will harvest on that to make the high level decision.

So during the process we reuse pre-trained models and layers.
There are multiple ways to go.

- We can completely freeze them and not train them.
- We can train some of them
- We can train the full new architecture



Outcomes:

A tool that allows the detection and amendment of personal data in text and association with information coming from *who is who*.

PoC 2: AI-based Search engine on keyword(s)

The user of the engine would type a selection of keywords or one keyword, the algorithm would then use these selected keywords to score all documents and return a list of the top "n" best matches of these documents ("n" to be specified by the user). The top "n" best matching documents will then be further classified by most occurring topics among them.

The Document types used by the search engine

All words and PDF documents plus email messages (".msg") in English will be analysed and scored.

Optional: we let the user select the library or folder for the search. For example, the user may only be interested in a subfolder/ part within ARES.

The keywords used for the search

The analyst will specify one or more keywords for the search. The more specific the keywords the more accurate the results will be. Generic terms like "Europe" or ,"Commission"…are not likely to provide specific results.
In order to make sure we cover the broadest semantic spectrum, the keywords are automatically matched with their semantically closest words (a method called word-embeddings will be applied here, see POC proposal 1). These closely related / synonymous words will then also be included in the search algorithm.

Note that the semantic matching of keywords will be ARES-based as the AI analysis (word embeddings) that will compute the semantic proximity of words is using ARES documents only. This will make sure that we fit to your specific semantic universe to find similar words.

Optional: We could also let the user specify the number of semantic matching words they would like to use. Note that increasing the number of retrieved synonyms will augment the chance of finding the right documents but it increases also the risk of retrieving documents that are not related at all to the initial goal of the search. Indeed the top 3 closest synonyms should be very close in meaning to the actual keyword selected by the user while the 7th or 10th closest synonym might be more distant in meaning.

The top "n" list of best matching documents

The algorithm will then "read" and score every document and give it a matching score. The scoring will be based on the selected keywords plus the 3 most similar words (semantically matched based on word embeddings) to each of the keywords. The algorithm will retain the top 100 best matching documents. The list appears in descending order of the matching score.

Optional: we could specify a parameter to be adjusted by the user so that you choose the amount of best documents returned as a result (top 25, top 50, top 75).

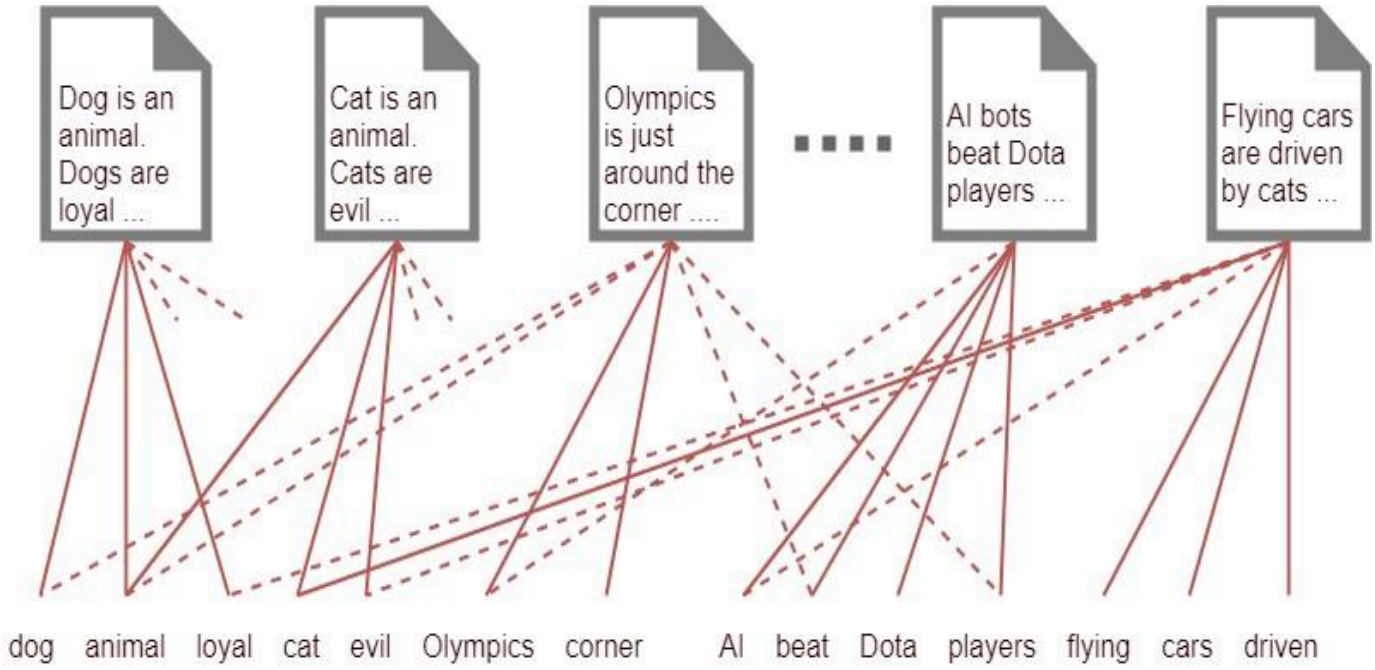
Classification of the top 100 into classes of most important topics found

The list of 100 documents will then be further analysed to find the most important "naturally" occurring topics covered within the top 100 documents (we will apply a topic modelling algorithm here). Each document is then assigned to a unique topic category. Naturally occurring topics (clusters) means that the algorithm will define the topics based on most occurring terms it finds in the 100 documents. Topics are not predefined by the user.

Optional: we can set a limit to the maximum number of topics to be considered.

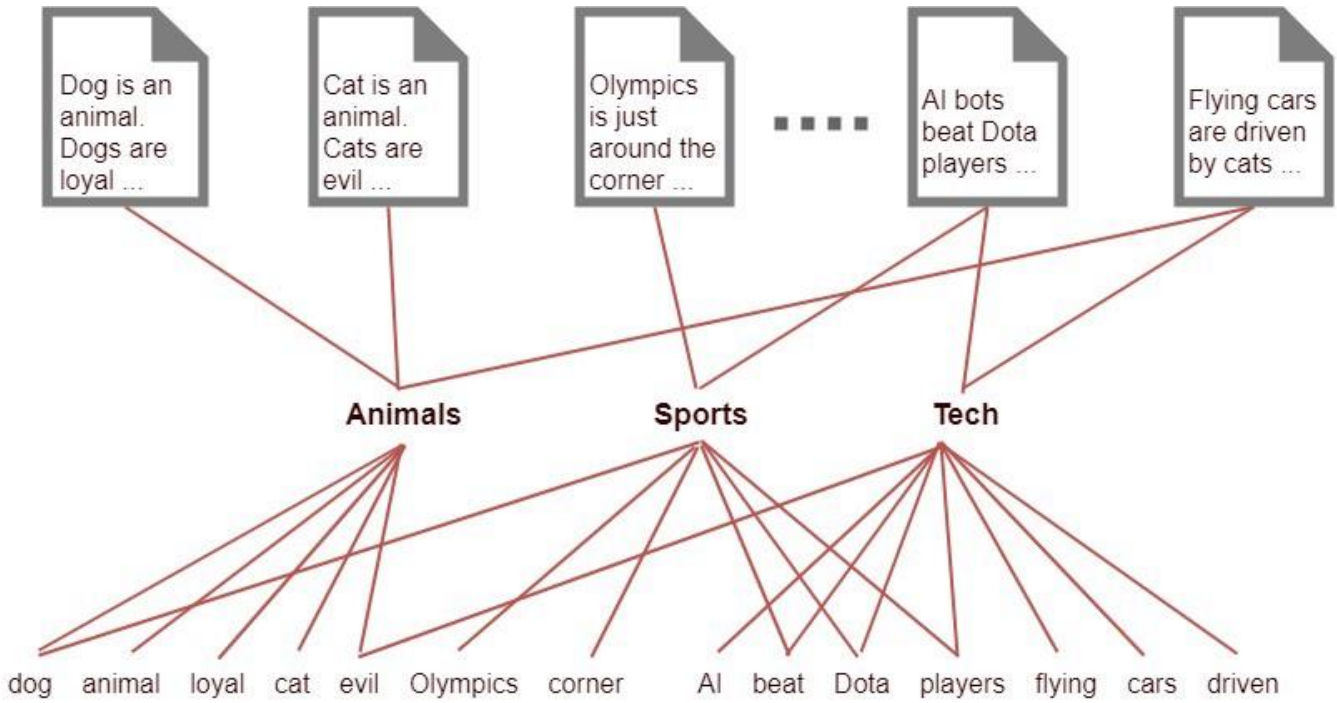
Topic modeling: a brief introduction:

Say you have a set of 1000 words (i.e. most common 1000 words found in all the documents) and you have 1000 documents. Assume that each document on average has 500 of these words appearing in each. How can you understand what category each document belongs to? One way is to connect each document to each word by a thread based on their appearance in the document. Something like below.



Modeling documents just with words. You can see that we can't really infer any useful information due to the large amount of connections

How can you solve this problem? By finding the latent "topics" class hidden in the data.



The algorithm will find the hidden topics (in this example: Animals, Sports, Tech). The final step will then be to assign each doc to its most relevant topics class.

The result / output from a search

The result will show for each class of topic found, the list of documents that are contained in this class rank ordered by their matching score to the keywords used in the Analysis.

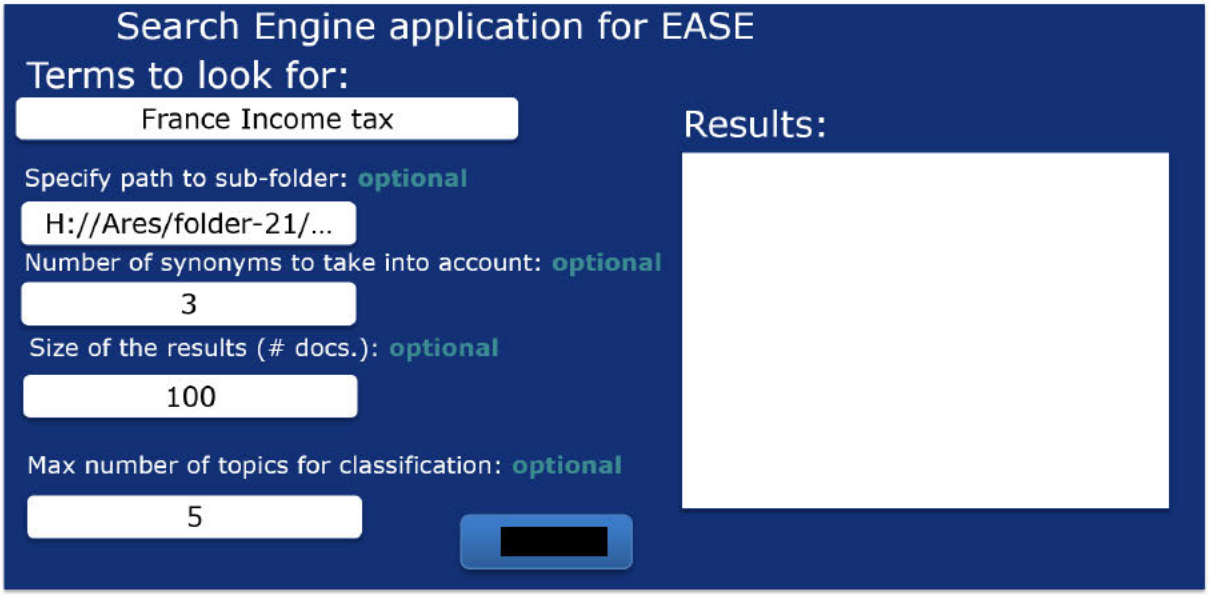
User scenario

Francine wants to retrieve all specific documents that relate to “French income tax”

1st step: Francine defines her search

She types in the keywords she thinks are the most relevant to detect what she is looking for.
Then she can decide to modify the optional parameters:

- A sub-folder to restrict the search to only one area of the library
- The number of synonyms to the her keywords that she wants to add to the search. She could increase this number if she wants to minimize the chances of missing a document, however by doing so she might also get more documents of a lesser relevance.
- She can decide to retrieve more or less than 100 documents in the final output.
- She can decide that she wants to see results for more or less than 5 topics in the total number of documents

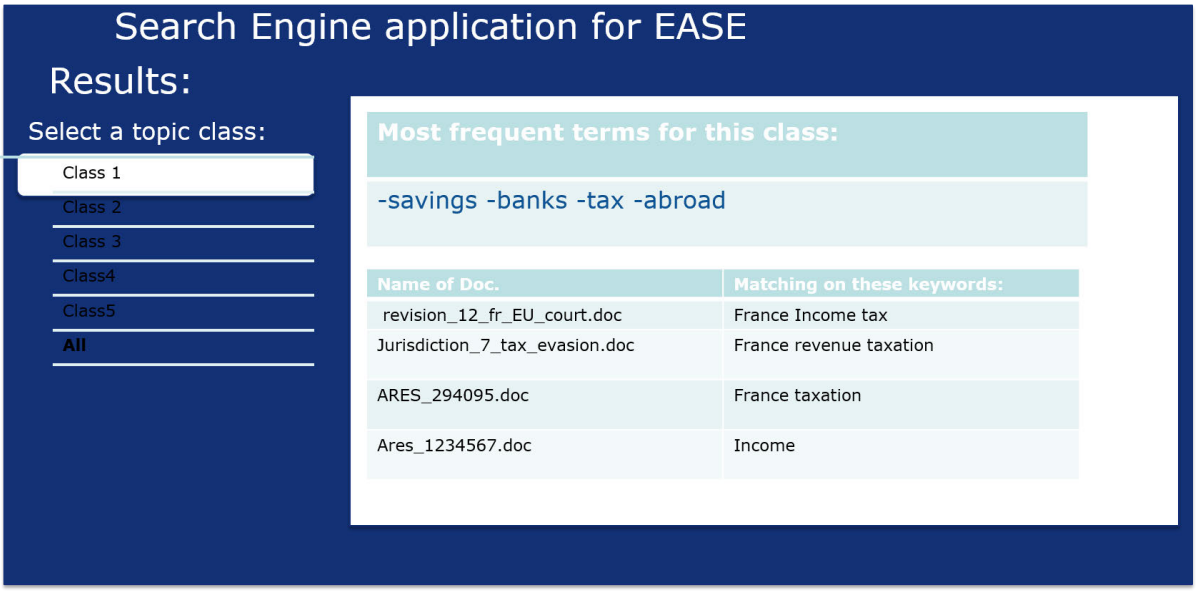


2nd step: after hitting the RUN button she lets the algorithm do its job:

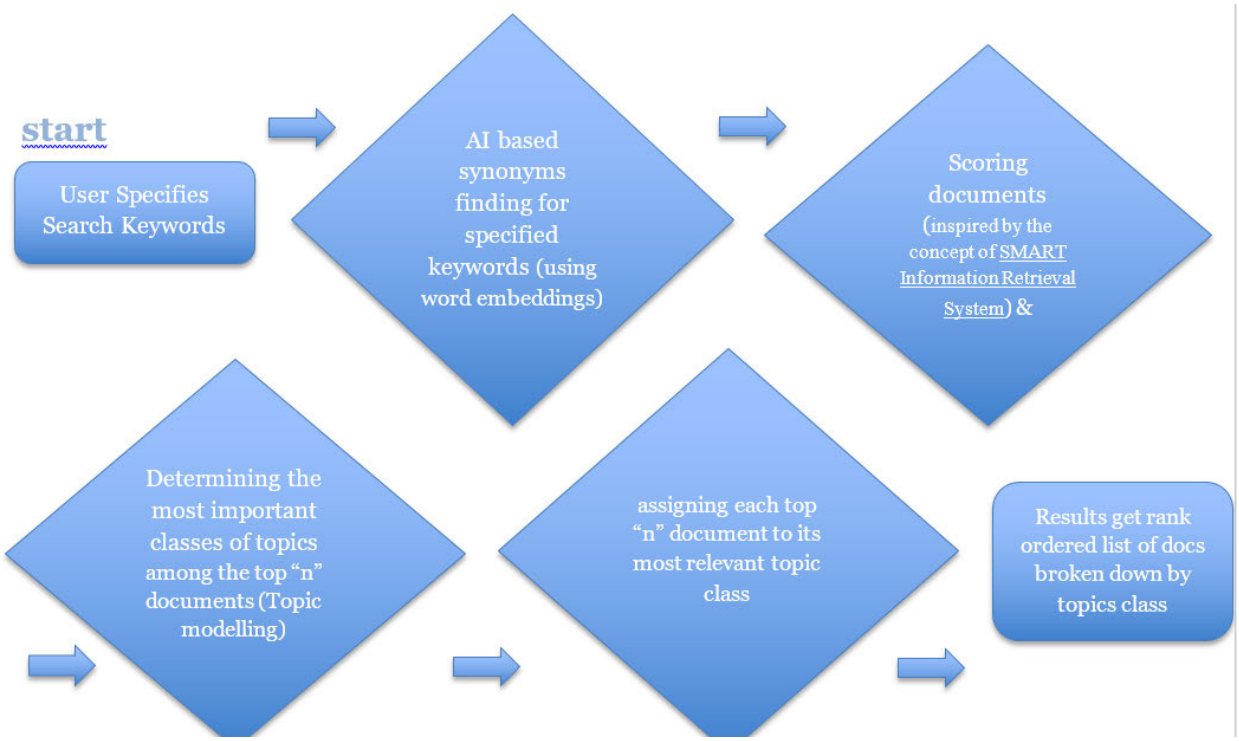
1. Collect the docs
2. Find the top 3 synonyms per keyword specified using word embeddings trained on your data (optional: the user specifies the number of synonyms)
3. Select the top 100 best matching with the keywords (including synonyms, optional: the user sets the number of top documents to be returned),differents methods could be chosen here to do the job: for example inspired by the concept of SMART Information Retrieval System or another example an Elastic search based method.
4. Analyse the 100 top documents to find the 5 most frequent topic classes (optional: user specifies number of topic classes), the method used here is Topic modelling.
5. Assign each of the top 100 docs to its closest topic class

3rd step she checks the outputs:

She can choose the topic class that she wants to investigate/ understand and receive the list of most common terms in the topic. She also sees a window with the rank ordered list of best matching documents that belong to that topic.



The process at a glance:



PoC 3: Find similar previous requests

Using this interface the user will be able to find similar previously answered request. The interface identifies similar requests and shows them to the user. All requests are ordered using score similarity.

Why use A.I?

Artificial Intelligence's biggest advantage is its ability to learn and generalize rules from data. The learned rules are then applied on new unseen data for different purposes.

When working with documents the main challenges are:

- Large amount of data to be processed
- Repetitive tasks
- Different formats and Languages
- Duplicates of contents
- Different "styles" and context of content
- Quality
- Time to deliver

A.I has the ability to adapt to any particular "style" and context to extract rules.
These rules can be used to:

- Identify points of interest in the documents
- Identify similar parts of documents
- Identify similar documents
- Enforce 'style'
- Verify Quality
- Create or translate documents
- Provide a continuous and improving quality of delivery

By combining A.I with current state of the art computational power:

- Large amount of documents can be processed extremely fast
- Data can be processed 24/7 with equal quality of delivery

By combining A.I with human verification and interaction, systems that merge speed, accuracy, continuous quality with human abilities can be built. Moreover providing feedbacks to the A.I will allow the system to continuously evolve. By using this approach a so-called *human-in-the-loop systems* is built.

Step 1: Creation of an index

During this stage the requests, responses and attached documents will be indexed. Indexing will permit to retrieve similar requests fast without needing to analyse all the documents at every search.

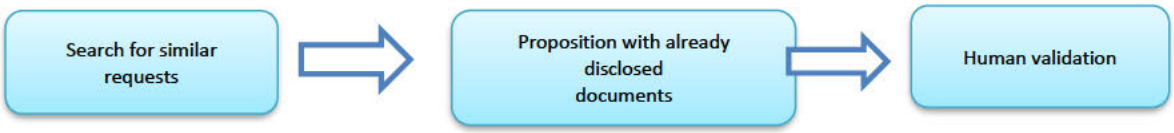
Step 2: Defining the best approach to calculate similarity

During this stage, the different approaches to calculate similarity between documents will be analysed and compared. The approach or the combination of approaches that provide the best combination of accuracy, performance and user friendliness will be choose.

Step 3: User interface

A simple interface that will permit the analysis of one or multiple files will be developed. For every request, the user will be able to visualize a list of previous requests ordered by similarity score accompanied by meta data as summary, topics, score etc.

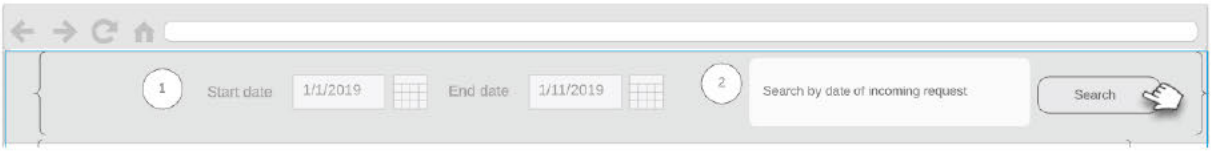
Workflow



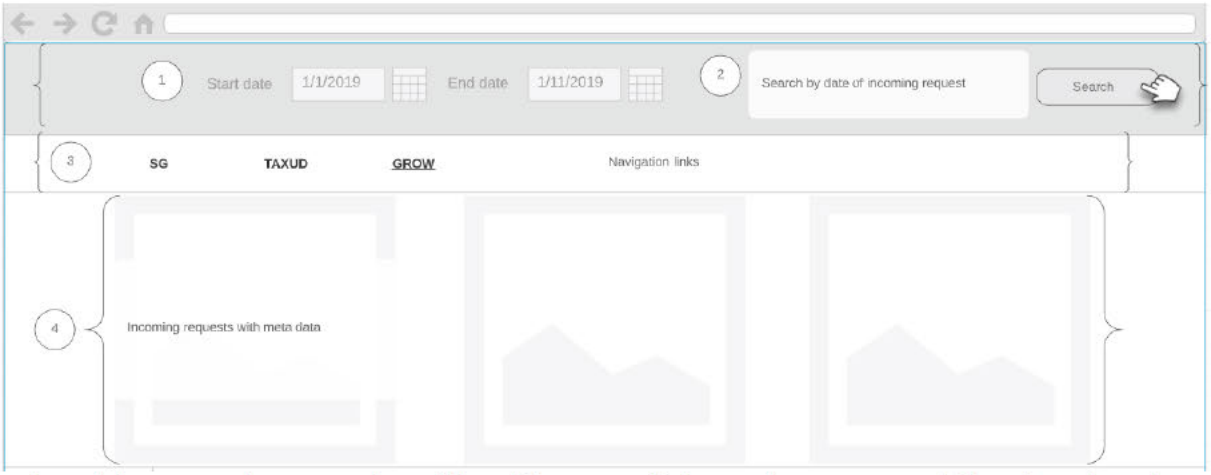
User scenario

Francine is in charge of dispatching incoming requests to the concerned DG's. It is not uncommon that the same request has been already answered and that the information can be disclosed without going through the complete workflow. To find similar previous requests Francine needs to manually search and read previous requests and responses to identify potential candidates.

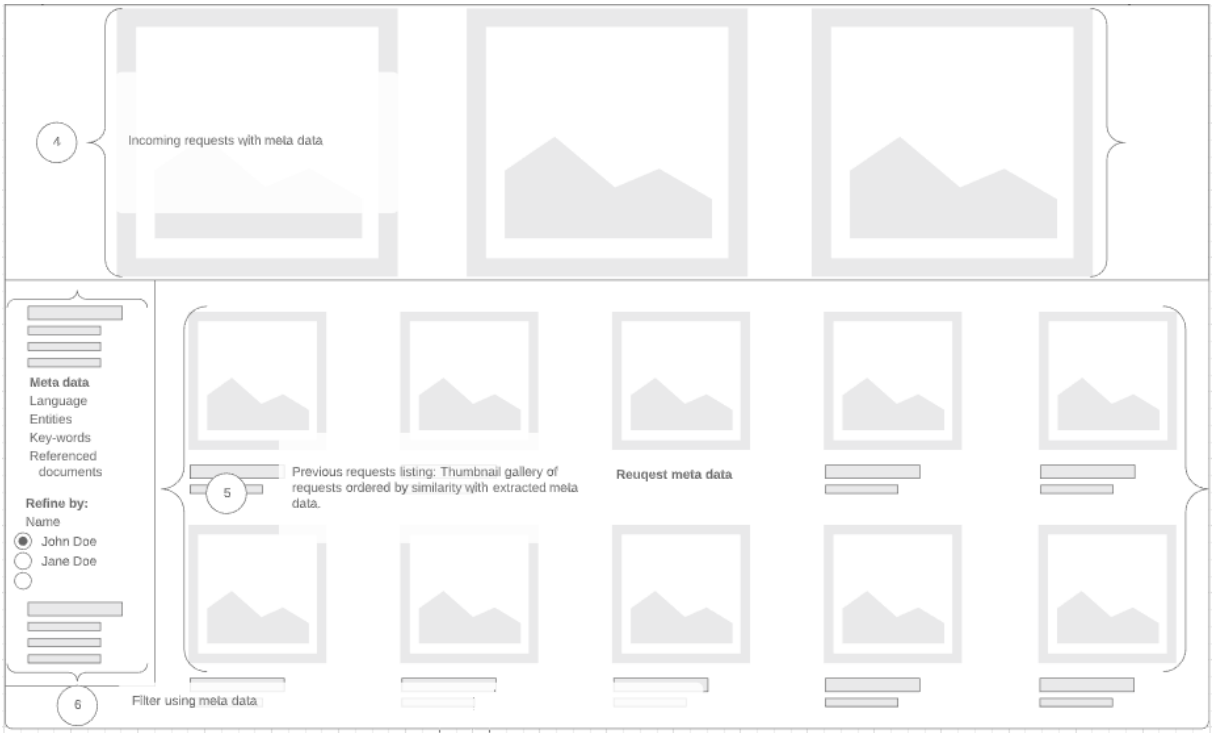
Instead doing this manually, Francine can use web interface that matches all the incoming requests to previously answered requests. She can select a time range and visualise all the incoming requests.



She will be presented a list of incoming requests with retrieved meta-data. Francine will be able to navigate through the incoming requests by for example the destination DG of the request.



When clicking on a document she will be able to see all the similar request and filter them based on meta data.



Francine is now able to afectivly analyse and search within multiple requests and responses to find a potential match.

Technology

To achieve the above-mentioned functionalities documents need to be effectively compared to each other. In order to compare the documents different approaches can be investigated.

Using embeddings:

When working with words and letters, these words and letters needs to be fed into the algorithm in order to transform them into arrays of numbers. The same method is applied with images. In order to achieve this, a vocabulary where every word is given a unique key is created.

Let's say we have a vocabulary of 3 words. Each of them has a unique number.

Rome → 1, Paris → 2, Italy → 3

At this point a one-hot encoding can be created to represent these words. One hot encoding represents a vector (an array) that has the length of the vocabulary's size. Meaning if there is a four words vocabulary each words will have an array of size four [0,0,0,0]. The word represented is written by using a '1' at the index of the word in the vocabulary.
For example:

Rome = [1,0,0,0,0]
I = [0,1,0,0,0]
Went = [0,0,1,0,0]
To = [0,0,0,1,0]
Italy = [0,0,0,0,1]

At this point a sentence (i.e. a sequence of words) can be represented by using vectors (arrays). The sentence: 'I went to Rome' can be represented as a sequence of one hot encodings as follow:

[[0,1,0,0,0] , [0,0,1,0,0] , [0,0,0,1,0] , [1,0,0,0,0]]

This representation is then fed into an algorithm. However it is not an ideal representation. The vectors (arrays) contain mostly zeroes (sparse) meaning that a lot of data will be needed in order to train an algorithm.

Given two sentences 'I went to Rome' and 'I went to Italy'. The difference between them does not represent the relationship between Rome and Italy. The indices at which the words Rome [1,0,0,0,0] and Italy [0,0,0,0,1] are saved in the vocabulary are random and do not represent any relationship. To solve this issue an approach called word2vec is used. In this approach the word as well at its context are looked at. One technique to generate word vectors (embedding) is Skip-gram.

Skip-gram:

Skip-gram looks at the context in which a word appears by looking at the words surrounding this word.
For example given the following sentence:

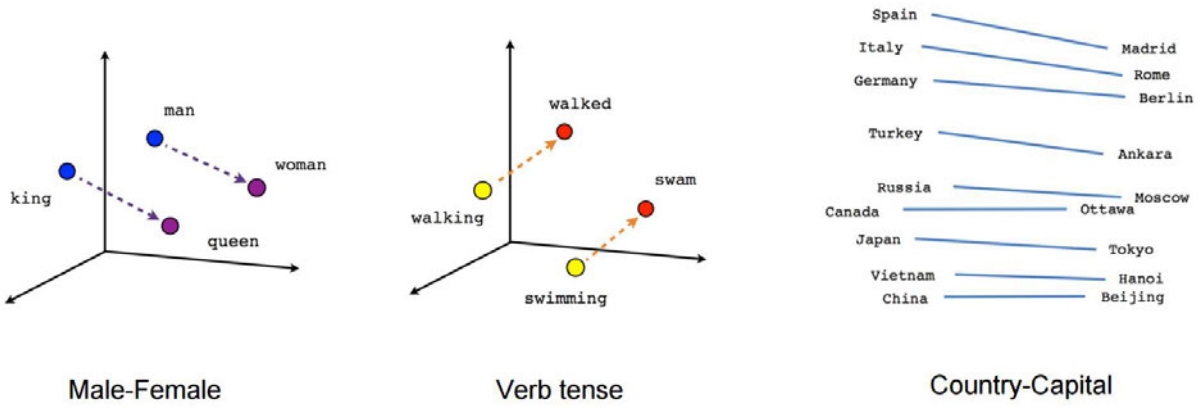
'I went to Rome'

The following pairs will be produced:

([I,to], went) , ([went,Rome],to)

These sequences can now be fed into a simple algorithm and trained using randomly generated negative samples (invalid sequences) and real sequences. The output of this algorithm is a dense vector (multi-dimensional array) called embedding. This is called semi-supervised learning.

These embeddings are able to capture the relationships between words and when computing the distance (difference) between them the relationship is represented.



Male-Female Verb tense Country-Capital

Skip-gram is not the only approach that can provide us with these kind of embeddings (context insights). Other possible approaches such as Glove and Bert are possible.

In this case we are looking for not only one word but a sequence of words and we need to be able to compare them.

The challenge is that when we deal text of variable length we can not effectively compare, calculate the distance, between two documents if the number of words in the documents are not the same.

The collection of embeddings generated by the sequences of words in the documents are wat we call high- dimensional data.
High-dimensional data often has properties that we can exploit. For example, high-dimensional data is often overcomplete, i.e., many dimensions are redundant and can be explained by a combination of other

dimensions.

Furthermore, dimensions in high-dimensional data are often correlated so that the data possesses an intrinsic lower-dimensional structure. To be able to effectively compare documents we can try to reduce the dimensions by using dimensionality reduction.

Dimensionality reduction exploits structure and correlation and allows us to work with a more compact representation of the data, ideally without losing information. We can think of dimensionality reduction as a compression technique, similar to jpeg or mp3, which are compression algorithms for images and music.

Currently there are a number of well documented and implemented dimensionality reduction algorithms that we can use:

Dimensionality Reduction with Principal Component Analysis (PCA)

In PCA, we are interested in finding projections of data points that are as similar to the original data points as possible, but which have a significantly lower intrinsic dimensionality.

t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-distributed Stochastic Neighbor Embedding (t-SNE) is a machine learning algorithm developed by [redacted] and [redacted]. It is a nonlinear dimensionality reduction technique well-suited for embedding high-dimensional data in a low-dimensional space.

By using one of the above mentioned methods whe can reduce the dimension of the collections to one where we can compare them to each other using cosine similarity or eclidian distance.

Letent Space

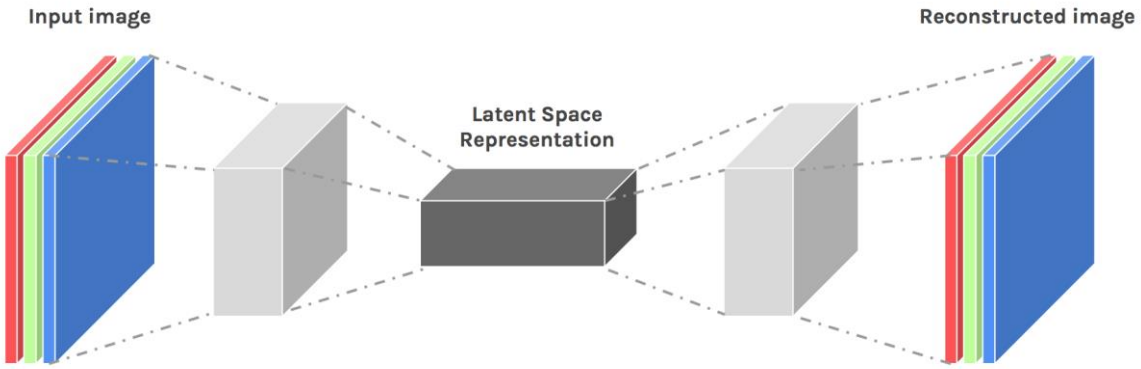
Another aproach to achieve the same functionalities is to leverage on unsupervised learning and a quite popular techinque to define extract latent features. Feature selection has been the topic of many studies. The basic intuition behind it is that we are looking to find elements in the text that define the document and give us the ability to find other documents with the same carectarstics.
This has proven to be quite difficult to do by a human. There are algorithms as XGboost that permit the selection of such features (feature importance).

Xboost

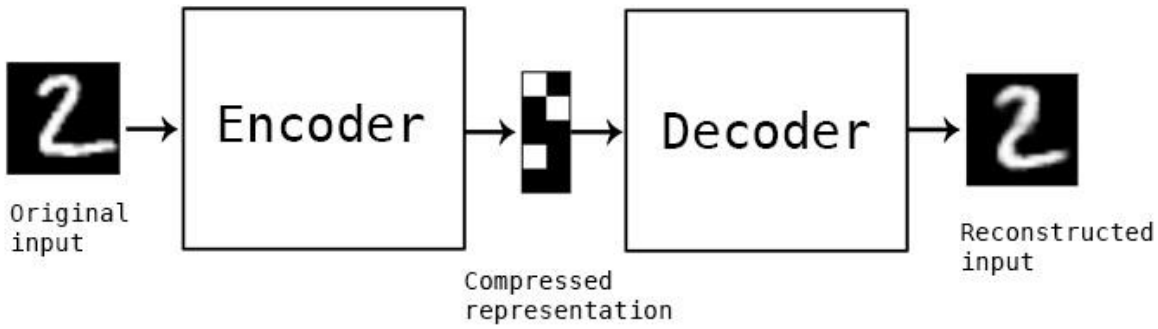
XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. A benefit of using ensembles of decision tree methods like gradient boosting is that they can automatically provide estimates of feature importance

Auto-encoders

The most popular and intuitive way to extract the latent features is to use auto-encoders.
Auto-encoders are artificial neural networks capable of learning efficient representations of the input data, without any supervision (the training set is unlabelled). More importantly, autoencoders act as powerful feature detectors, and they can be used for unsupervised pretraining of deep neural networks.



Autonencoders consist of an encoder and a decoder. The encoder takes in the input and runs it through a number of layers to arrive at a dense tensor that represents the latent space. The decoder takes this tensor as an input and its task is to regenerate the input that the encoder has seen.



By doing this we force the autoencoder to learn the latent features of the item without human intervention. This compressed representation can be used to compare objects similar to the word embeddings example. It has been proven that the compressed representation achieves a similar and some-times better dimensionality reduction and feature selection than the above mentioned approaches.

Outcomes

Tool that allows the analyses and selection of requests. The tool permits to find previous responses to a similar request.

PoC 4: AI generated responses

The user will be able to partially generate the text of a response based on a request.

Why use A.I?

Artificial Intelligence's biggest advantage is its ability to learn and generalize rules from data. The learned rules are then applied on new unseen data for different purposes.

When working with documents the main challenges usually are:

- Large amount of data to be processed
- Repetitive tasks
- Different formats and Languages
- Duplicates of contents
- Different "styles" and context of content
- Quality
- Time to deliver

A.I has the ability to adapt to any particular "style" and context to extract rules.
These rules can be used to:

- Identify points of interest in the documents
- Identify similar parts of documents
- Identify similar documents
- Enforce 'style'
- Verify Quality
- Create or translate documents
- Provide a continuous and improving quality of delivery

By combining A.I with current state of the art computational power:

- Large amount of documents can be processed extremely fast
- Data can be processed 24/7 with equal quality of delivery

By combining A.I with human verification and interaction, systems that merge speed, accuracy, continuous quality with human abilities can be built. Moreover providing feedbacks to the A.I will allow the system to continuously evolve. By using this approach a so-called *human-in-the-loop systems* is built.

Step 1: Create a dataset

During this stage a dataset that matches requests with their respective responses is created.

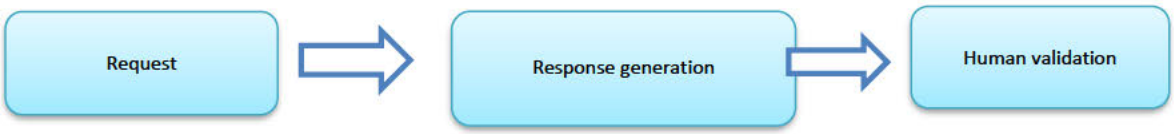
Step 2: Define the best approach to generate responses

During this stage, we will analyse and compare the different approaches to generate responses. We will choose the approach or the combination of approaches that provide the best combination of accuracy, performance and user friendliness.

Step 3: User interface

We will develop a simple interface that will permit the user to select a request and to ask the application to generate a number (variants) of responses.

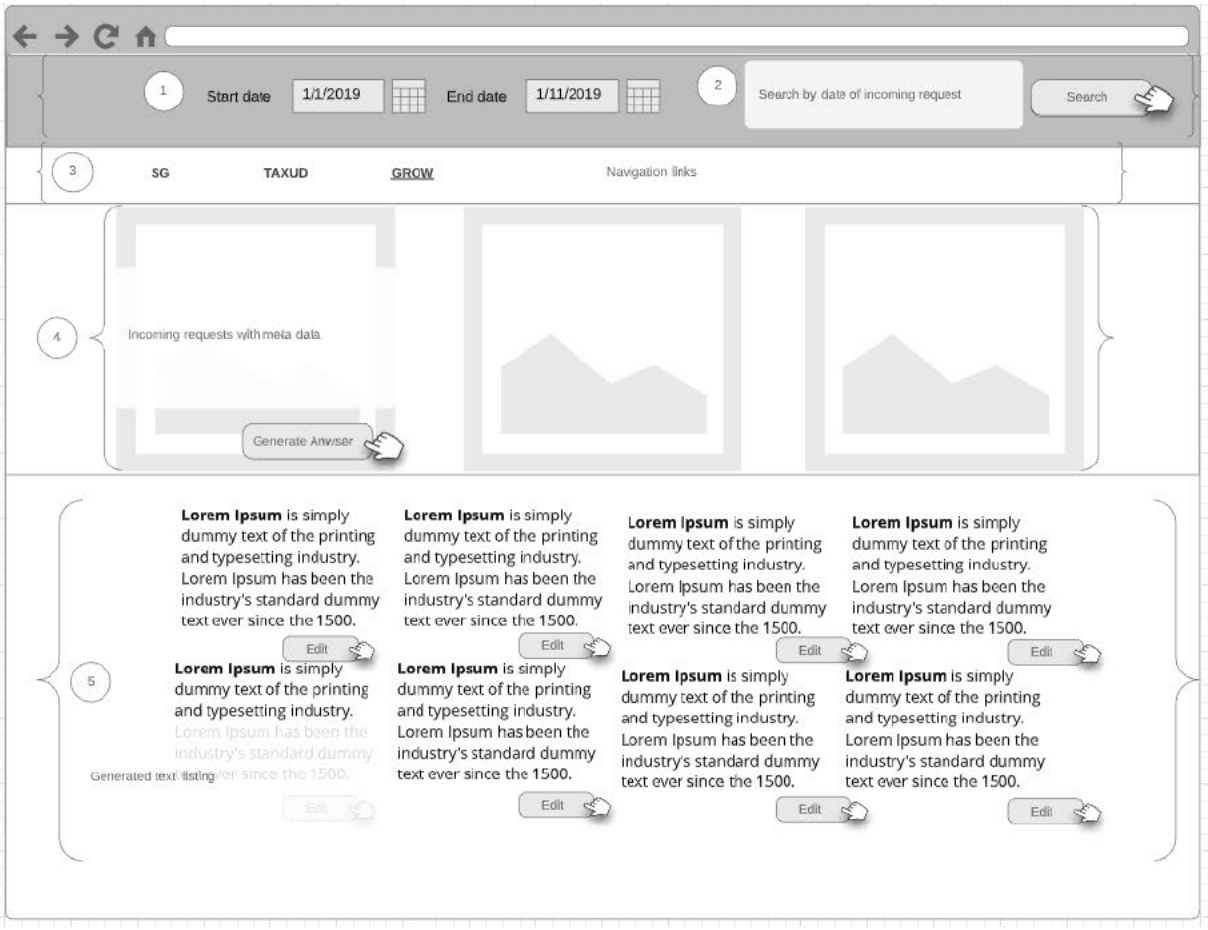
Workflow



User scenario

Francine is in charge of writing the response letters that are sent together with the documents to the citizens. Francine finds herself duplicating large portions of the text and only changing specifics related to the request.

Instead doing this manually, Francine can use web interface where she can select a request and ask the application to propose her different variants of a possible response.



Francine can visualize and edit the proposed text.

Technology

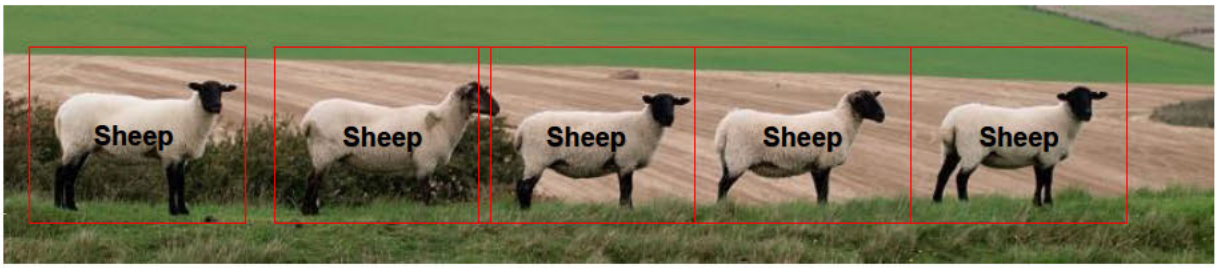
To achieve the above-mentioned functionalities we need to be able use the information from the request and generate a response based on that.

Using embeddings:

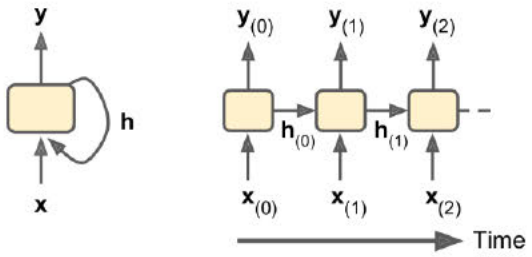
When we want to work with words and letters we use an approach called word2vec. We look not only at the word but as well at its context. Popular approaches for word2vec are Skip-gram, Glove and Bert.

In our case we are looking for not only one word but a sequence of words. Most algorithms do not have a memory and are not able to take previous inputs into account to generate a prediction.

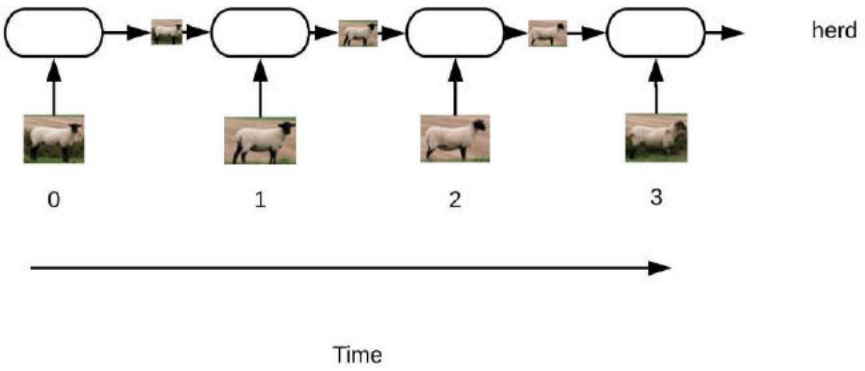
For example, an image classification algorithm can classify ten sheep, one after the other but it will not classify them as a herd of sheep.



An algorithm that can consider sequences and understand that seeing five sheep in a sequence means that it is looking at a herd is needed.
These algorithms are called recurrent neural networks RNN. And what distinguishes them from the rest is that they take two inputs one of them being the previous output.



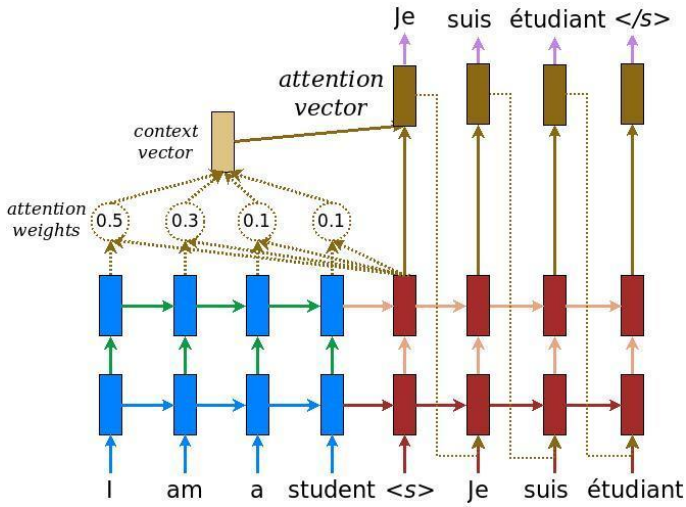
Using the sheep example:



In our case we want to take into account the request information and based on that generate a new sequence or words that will be the response.

Sequence to Sequence

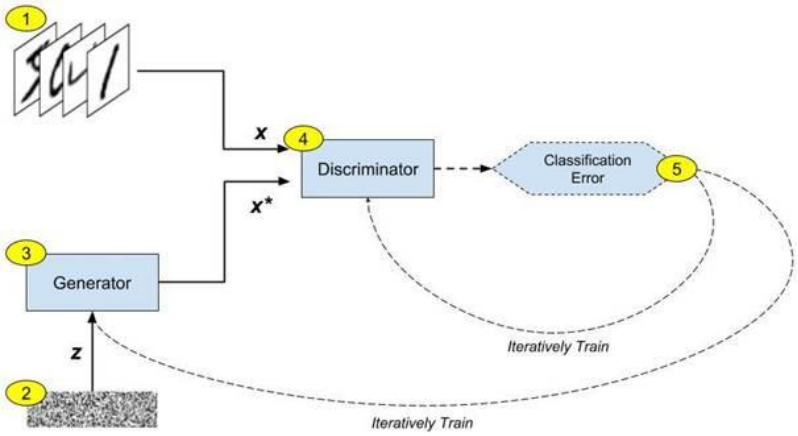
The most popular approach is the sequence to sequence architecture where the sequence of words (input) are fed into a deep learning algorithm called an Encoder. The encoder is Recurrent Neural Network. The output that the encoder generates is fed into a second RNN that is called the decoder. The decoder's task is to take these outputs and generate a new sequence.



This approach is usually used for machine translation where the inputs of the encoder are the sequence of words in one language and the sequence of words generated by the decoder are the equivalent in another language.

Generative Adversarial Networks

GAN is a battle between two adversaries—the generator and the discriminator. The generator tries to convert random noise into observations that look as if they come from the original dataset and the discriminator tries to decide whether an observation comes from the original dataset or is one of the generator's forgeries.



By competing against each other the generator becomes better in generating realistic samples and the discriminator becomes better in recognizing fakes from real. When they reach equilibrium the generator is able to generate samples that are stunningly close to real data.

GANs have achieved imagination-capturing results. Example generated faces by one of the latest GAN implementation **ProGAN's** using progressive growing.

Images generated by ProGan

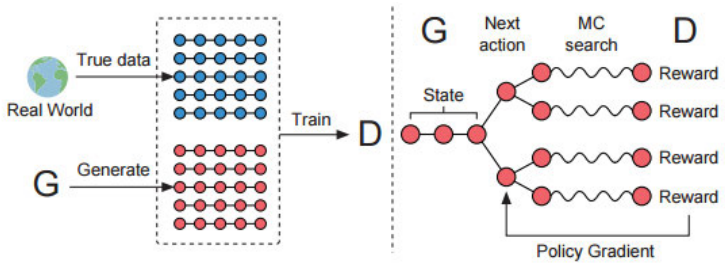
Stacked GAN's are a good example where GAN's can be used to take a sequence of words extract semantic information from the text and generate and output that represents closely the description given in the output.

Images generated by Stacked GAN

In terms of generating sequences of words there are a number of suitable candidates with in the GAN universe.

SeqGan:

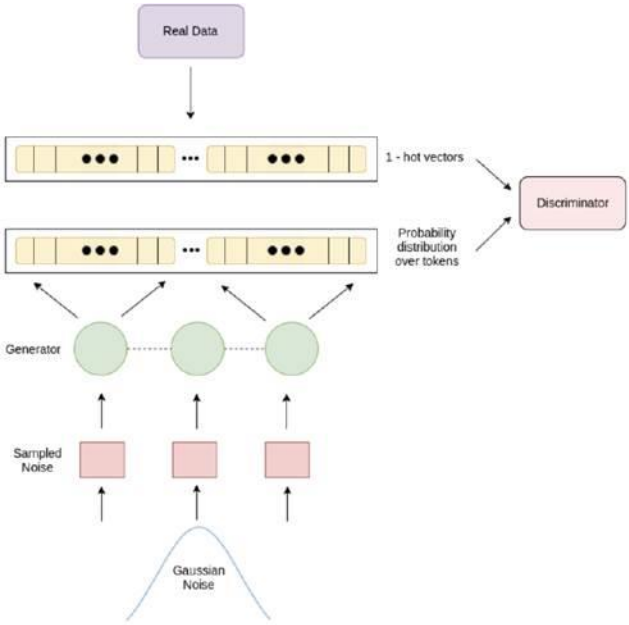
Left: Discriminator is trained over the real data and the generated data by the Generator.
Right: Generator is trained by policy gradient where the final reward signal is provided by D and is passed back to the intermediate action value via Monte Carlo search.



SeqGan is the base of ORGAN a project used for molecule generation.

Adversarial Generation of Natural Language

The paper Natural Language Generation with GANs by et al. investigates different GANs and their uses in sequence generation. The authors circumvent the problem of discrete generator output by using the distribution output. The authors were able to train GAN-based text-generation models using both RNNs and Convolutions.



Outcomes

Tool that allows the fast generation of text taking into account information retrieved from request. The tool provides the ability to quickly evaluate multiple possible versions of text that can be edited and validated by a human.