Document Type: #draft
Category:
Parent:

Notes

12.04.2024 22:35
I am going to use my knowledge and searching skill to develop a MVP for a chatbot.
As my master thesis is in news & topics analysis domain, I will follow the same course.
I will document all my train of thought and GPT Prompting I will be using to reach the final result.
First, I need to properly create the requirements;

Step 1 Requirements

Result: API REST endpoint provides data in a json format. Each document consists of a category, news headline body text and timestamp. The language chosen is Romanian. At first, data is raw so processing techniques should be applied. After the data is formatted apply a topic algorithm to extract topics discussed for each document. The result should be a valuable insight about what are the most common topics discussed over time.

(**12.04.2024 22:45**) LangChain has been on my watchlist to understand. I will use this project as an opportunity to grasp what exactly it is doing and find an usecase for it.

(It seems there are other valuable alternatives such as Microsoft AutoGen compared to LangChain)

Looking first if I could use OpenAI GPT to actually preprocess the text. One of the problem is that the text is romanian, so I need to find a good enough technique to obtain good quality data.

At first sight, I feel like #LangChain is just an interface for LLMs.

Business Requirements

> (**12.04.2024 23:25**) Using drafts/CRISP-DM process model to define the process of my project.

1. Business understanding
   Local newspaper requires to design a system that would allow the user to have a bird's-eye view on the topics discussed in the news. Raw data extracted from the news should be processed and algorithms should be used to extract relevant topics. Various visualizations should be used to show the evolution of discussed topics over the period of the obtained time frame.
2. Data understanding
   Data consists of raw news article.
   The main language is Romanian.
   The data consists of a headline, body text, timestamp and its manual labelled category.
3. Data preparation
   Reprocessing techniques should be apply to make the data appropriate for topic modelling.
   I should first remove the stop words, apply NER techniques to identify names, companies and location and normalize the text (stemming).
   Each document should consist of a final formatted form of:

```json
{
id: id
headline: "Title",
timestamo: DD-MM-YYYY HH:MM
category: ['exemplu']
content: ['acesta', 'a fost', 'exemplu']
}
```

4. Modeling
   LDA and NMF topic algorithms should be used. Numbers of topic

should vary between 1 and 5 topics using an automated technique to choose the best option using coherence score analysis.

5. Evaluation
   ################
6. Deployment
   ################

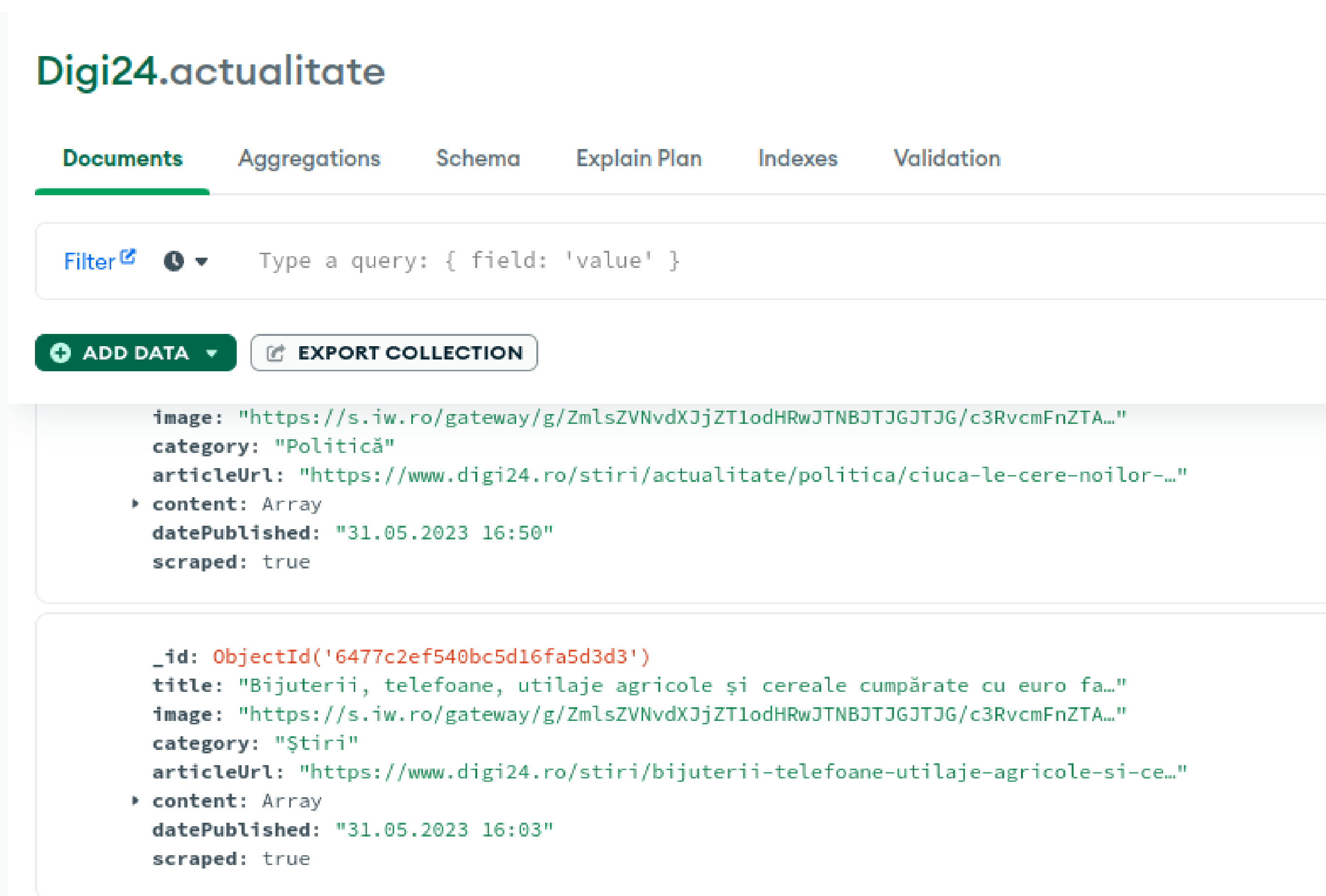
Step 2 Project Structure

I should define first an architecture using a cookie cutter project. Based on what I've read before, I will use the following:

> Using Data Science Cookiecutter project structure.

> Will use Pipenv as it is also used by the Tremend team.

Since I already have some news data stored somewhere on my computer, I will create a #docker container with mongodb and the data from where I can directly fetch the data.



```dockerfile
version: '3'
services:
        containerNews:
        image: docker.io/bitnami/mongodb:7.0
        ports:
                - "27020:27020"
        environment:
                - MONGODB_PORT_NUMBER=27020
        volumes:
                - 'mongodb_data:/bitnami/mongodb'
                - './data:/data'
volumes:
        mongodb_data:
                driver: local
```

Creating now an `EXTRACT` script pipeline to get the data from the Container on local.

(**12.04.2024 01:15)** closing for now

13.04.2024 11:15) starting, 15:33h left

I first need to create a notebook to experiment how to deal with preprocessing the data.

Created a dataframe with the structure I need.

	category	content	datePublished	title
0	Politică	["Viceliderul grupului deputaţilor UDMR Szabo ...	31.05.2023 23:29	Deputat UDMR: Trebuie reconstruită încrederea ...
2	Actualitate	["Staţia de metrou Piaţa Iancului este prima s...	31.05.2023 22:35	Metrorex a anunţat care este prima staţie de m...
3	SUA	["Fostul vicepreşedinte republican Mike Pence ...	31.05.2023 22:34	Fostul vicepreședinte al SUA Mike Pence se pre...
4	Economie	["Horia Constantinescu, preşedintele Autorităţ...	31.05.2023 21:34	Şeful ANPC, despre inspectorii care iau şpăgi:...
5	Educaţie	["Guvernul anunţă într-un comunicat de presă e...	31.05.2023 20:53	Guvernul insistă cu oferta respinsă de profesori
...	...	...	...	...
9992	Social	["O tânără în vârstă de 19 ani şi o minoră în ...	07.10.2022 10:37	Două tinere au furat produse de 2.100 de lei d...
9993	Externe	["Statele Unite ale Americii sunt „pregătite” ...	07.10.2022 10:36	Blinken: SUA sunt pregătite pentru o soluție d...
9994	Externe	["Vicepremierul Republicii Moldova Andrei Spîn...	07.10.2022 09:18	Republica Moldova așteaptă gazele din România....
9995	Social	["Retailerul Lidl România a anunţat că retrage...	07.10.2022 08:25	Produse cu Salmonella Typhimurium la Lidl. Cli...
9996	Politică	["Aflată în zona centrală a Africii, Rwanda a ...	07.10.2022 08:00	Deplasări all inclusive pentru parlamentarii r...

9200 rows × 4 columns

I will invest exactly 1 hour to find some interesting methods of preprocessing my text. I belive using a Transformer to identify NER would be interesting.

I have a problem with spacy, i get "no module named" - It worked, it was actually a problem of absolute paths. I needed to isntall spacy in the same location as my notebook.

Downloading ## [ro_core_news_lg]() to see some results
Still have problems with submodules - I opened jupyter in the wrong place.
`spacy download ro_core_news_lg`

I created a very simple pipeline that removes stopwords and lemmatize the words. I am now looking at a good ideea for NER.

`fost vicepresedinte republican Mike Pence pregateste 7 iunie cursa alegere prezidential 2024`

I got the following ideea- I should use ro_core_news_lg and create tokens based on NER with the following simple rule: If there are multiple succesive same-type entities, I should combine together (example: names, dates, organisations)

`['fost', 'vicepresedinte', 'republican', 'Mike Pence', 'pregateste', '7 iunie', 'cursa', 'alegere', 'prezidential', '2024']`

I still believe [bert-base-romanian-ner]() is a great try, but i need to continue finishing this mockup and then return for more.

(13.04.2024 14:00) Almost forgot about git flow in my workflow - I experimented with "sketchbook" and now i should create a new notebook, naming it "text_process_01.ipynb" to create the basic pipeline example. I will also leave sketchbook alone for now since i have some code for transformers i might need later.

waiting for more than almost 7 minutes and the pipeline did not finish. **#todo** ☑️ I guess it's proper time to understand what options do I have to parallize this, takes too much - I need to research what options i have - my intuition is that i could use Spark.

```
df["processed_content"] = df["content"].apply(process_pipeline)
[13]  6m 11.9s
```

Creating a branch `experiment/text_process_01` to save this form and for now i will work only with 10 first rows.

Creating a new branch `experiment/topic_modeling_01` where i can experiment some results from my process.

Evaluating gensim, forgot some key aspects.
So documents were text sequence, corpus were collection of documents.
Latent means hidden, yet to discovered.
Unsupeversied lantent nature of the model require me to represent the documents in vectors of features.
A good example offered by the documentation is placing a set of questions which result in dense vectors:

1. how many times word X appears -> dense vector (x1,y1)
2. how many times paragraphs does the document of (x2,y2)
   and then compare two documents based on the pairs (x1,y1) with (x2,y2) to conclude the answer.

Remember that I get sparse vectors because each tokenize represents a dimension in the vector, so nxn.
remember that documents exist in document space, and that vectors exist in vector space, the above ambiguity is acceptable.

doc2bow counts occurences of distinct word and convert them into sparse vector

Gensim accepts any iterable object (iterators)

Gensim transformations I could find useful:

- draft/tfidf - rare features locally and globally relevance

- okapi bm25 - Similar to tfidf but takes into account sentence length and saturated results (too many common words)
- LSI - transforms bag-of-word of tf-idf to lower dimensionality latent space using SVD. (200 - 500 dimension recommended)
- Random Projections - memorhy cpu efficient to reduce vector space dimensionality.
- LDA --> POINT OF Interest
  bag-of-words into lower dimensionality topic space.

Why LSA is named multi-polynomial PCA
I believe because there are multiple dimensions for each pair of words, and PCA is a reduction techinique to visualize similarity.

#bookmark I find it useful that LSI training accepts continue training.
This should help me since my data lineage should have as and result and infinite stream of documents fro mdaily news.
An important note is that i can tweak the features for the data stream, for example i can set a "forget old observation" setting.
check here

https://issues.apache.org/jira/browse/SPARK-20082

```Python
model.add_documents(another_tfidf_corpus)  # now LSI has been trained on tfidf_corpus + another_tfidf_corpus
lsi_vec = model[tfidf_vec]  # convert some new document into the LSI space, without affecting the model
model.add_documents(more_documents)  # tfidf_corpus + another_tfidf_corpus + more_documents
lsi_vec = model[tfidf_vec]
```

topic drift
batch vs online
Some suggest it is better to batch train LDA instead of online since a topic drift would lower the quality of the model.

This article talks about ussing LLM to summarize pdf by first extracting topics using draft/LDA, which reduces costs by 99%
https://towardsdatascience.com/document-topic-extraction-with-

large-language-models-llm-and-the-latent-dirichlet-allocation-e4697e4dae87

I forgot i had a .gitkeep in my notebooks, i lost my two notebooks and the code i made so far. I am trying to find a solution to recover it.
There are no good solutions - Jupyter notebooks are not supposed to be version controlled :(.

trying to do a little more to the preprocess pipeline, using stanzaLanguage and spacy again.

created an even better pipeline thank to this fail, but I spent a lot of time.
this page helped me a lot: https://github.com/Alegzandra/Romanian-NLP-tools

```python
import pandas as pd

import re

import stanza

import spacy_stanza



nlp = spacy_stanza.load_pipeline("ro")



def lemmatize_tokens(tokens):

doc = nlp(" ".join(tokens))

lemmatized_tokens = [token.lemma_ for token in doc]

return lemmatized_tokens
```

```
def remove_ner(text):

doc = nlp(text)

tokens = []

for token in doc:

if token.ent_type_ not in ['MONEY', 'DATE', 'TIME',
'QUANTITY', 'ORDINAL', 'CARDINAL', 'NUMERIC_VALUE', 'PERSON',
'DATETIME']:

alpha_chars = [char for char in token.text if char.isalpha()]

cleaned_token = ''.join(alpha_chars)

if cleaned_token:

tokens.append(cleaned_token)

return tokens



def remove_stopwords(tokens):

stopwords = spacy.lang.ro.stop_words.STOP_WORDS

filtered_tokens = [token for token in tokens if token.lower()
not in stopwords]

return filtered_tokens



def preprocess_text(text):
```

```
tokens = remove_ner(text)

tokens = remove_stopwords(tokens)

tokens = lemmatize_tokens(tokens)

return tokens



def tokenize_text_from_csv(csv_file):

df = pd.read_csv(csv_file, usecols=['_id', 'category',
'datePublished', 'content'], nrows=10) # Read only the first
10 rows

tokenized_data = []

for index, row in df.iterrows():

content = row['content']

tokens = preprocess_text(content)

tokenized_data.append({

'_id': row['_id'],

'category': row['category'],

'datePublished': row['datePublished'],

'tokens': tokens

})

tokenized_df = pd.DataFrame(tokenized_data)
```

```
tokenized_df.to_csv("tokenized_output.csv", index=False,
columns=['_id', 'category', 'datePublished', 'tokens'])



# Example usage:

csv_file_path = "actualitate.csv"

tokenize_text_from_csv(csv_file_path)
```

coming back to topic_modeling to see how this handles now the data. creating branch `experiment/data_pipeline_01`

I need to use a score to automatically choose number of topics - Coherence score seems like a good ideea.

```
Optimal number of topics: 2
Topic 0: 'loc',, 'Ucraina',, 'România',, 'spune',, 'Rusia',, 'persoană',
Topic 1: 'sindicat',, 'guvern',, 'profesor',, 'leu',, 'grevă',, 'lider',
```

13.04.2024 23:25) I think a better approach would be to train an LDA for each category separately (politica, sport, etc)

Stopping for now. Even if the result is coherent, i think it's a bad ideea to create a topic for all of articles, instead I should train LDA on the entire data set and then create topic for each headline separately.

14.04.2024 12:50)
I first need to see why some parts of text won't feed the LDA.

`Error: cannot compute LDA over an empty collection (no terms). Skipping...`

Found a solution. I believe bag of words is not useful at all for my scenario. Trying to use tf-idf and then find other text representation techniques for my lda.

interesting article about using bert embeddings:
https://towardsdatascience.com/topic-modeling-with-bert-779f7db187e6

using bag-of-words, horrible result

```
Topic: 0
Words: ['loc', 'respinge', 'următor', 'reprezentant', 'sistem', 'Newsro', 'ac', 'urma']

Topic: 1
Words: ['loc', 'reprezentant', 'urma', 'ac', 'Newsro', 'sistem', 'următor', 'respinge']

Topic: 2
Words: ['sistem', 'respinge', 'urma', 'ac', 'următor', 'reprezentant', 'Newsro', 'loc']

Topic: 3
Words: ['sistem', 'loc', 'ac', 'Newsro', 'următor', 'urma', 'reprezentant', 'respinge']

Topic: 4
Words: ['respinge', 'Newsro', 'ac', 'următor', 'reprezentant', 'urma', 'loc', 'sistem']
```

Using pretrained Spacy ro_core_news_lg for its word embeddings got me some good results:

```
Topic: 0
Words: ['spune', 'economic', 'operator', 'sursă', 'pune', 'coleg', 'mână', 'constantinescu', 'minunat', 'încet']

Topic: 1
Words: ['locomotivă', 'tren', 'brașov', 'abur', 'fum', 'regal', 'an', 'transmite', 'loc', 'putea']

Topic: 2
Words: ['persoană', 'staţie', 'metrou', 'tate', 'monta', 'panou', 'bucureşti', 'proiect', 'element', 'informare']

Topic: 3
Words: ['medic', 'leu', 'minister', 'loc', 'guvern', 'familie', 'sindicat', 'an', 'medicină', 'brut']

Topic: 4
Words: ['leu', 'guvern', 'sindicat', 'an', 'majorare', 'brut', 'salariu', 'ofertă', 'grevă', 'educație']
```

Saving this as lda_04_spacy.py.

Now I want to create the data pipeline itself.
One problem I bookmarked was that the text processing was very slow - I need to find a way to parallelize the process.

checking out to 'experiment/spark_process_01' and using some time to document about how to do it.

knowledge/Apache Spark getting different spark errors such as `ValueError: [E1041] Expected a string, Doc, or bytes as input, but got: <class 'function'>`

fixed the error, but now i get an error caused by Stanza that cannot be parallized:

`TypeError: cannot pickle '_thread.lock' object`

Pyspark uses `PickleSerializer` to serialize the python objects, but spacy models aren't serializable using `PickleSerializer` which is trigger the issue when we load the spacy model first and then refer to worker code.

Found a solution, I modified the following:
move initialization of spacy inside the Function so each cluster has its own nlp object.

moving on to create pipeline for all functions

`stopwords = spacy.lang.ro.stop_words.STOP_WORDS`

seems the problem was again that i forgot to import to package inside UDFs

```python
def remove_stopwords(tokens):

import spacy

nlp = spacy_stanza.load_pipeline("ro")
```

It works!! Compared to the normal pipeline that took 25 minutes to process the data, I got it now in under 90 seconds.

```python
import findspark

findspark.init()


from pyspark.sql import SparkSession
```

```python
from pyspark.sql.functions import col, udf

from pyspark.sql.types import StringType



# Create a Spark session

spark = SparkSession.builder \

.appName("Spark Transformer") \

.getOrCreate()




csv_file = "actualitate.csv"



df_spark = spark.read \

.format("csv") \

.option("header", "true") \

.option("inferSchema", "true") \

.load(csv_file) \

.select("_id", "category", "datePublished", "content",
"title") \

.limit(10000)
```

```
def remove_ner_spark(text):

import spacy

nlp = spacy.load("ro_core_news_lg")

doc = nlp(text)

tokens = []

for token in doc:

if token.ent_type_ not in ['MONEY', 'DATE', 'TIME',
'QUANTITY', 'ORDINAL', 'CARDINAL', 'NUMERIC_VALUE', 'PERSON',
'DATETIME']:

alpha_chars = [char for char in token.text if char.isalpha()]

cleaned_token = ''.join(alpha_chars)

if cleaned_token:

tokens.append(cleaned_token)

return tokens



def remove_stopwords(tokens):

import spacy

stopwords = spacy.lang.ro.stop_words.STOP_WORDS

filtered_tokens = [token for token in tokens if token.lower()
not in stopwords]
```

```
return filtered_tokens


# Define the function to lemmatize tokens

def lemmatize_tokens(tokens):

import spacy

stopwords = spacy.lang.ro.stop_words.STOP_WORDS

nlp = spacy.load("ro_core_news_lg")

doc = nlp(" ".join(tokens))

lemmatized_tokens = [token.lemma_ for token in doc]

return lemmatized_tokens


# Register the functions as UDFs (User Defined Functions)

remove_ner_udf = udf(remove_ner_spark, StringType())

remove_stopwords_udf = udf(remove_stopwords, StringType())

lemmatize_tokens_udf = udf(lemmatize_tokens, StringType())


# Apply the UDFs to create new columns

df_processed = df_spark.withColumn("cleaned_tokens_ner",
remove_ner_udf(df_spark["content"])) \

.withColumn("cleaned_tokens_stopwords",
```

```
remove_stopwords_udf(col("cleaned_tokens_ner"))) \

.withColumn("cleaned_tokens_final",
lemmatize_tokens_udf(col("cleaned_tokens_stopwords")))



# Show the resulting DataFrame

df_processed.show(truncate=False)



# Stop the Spark session

spark.stop()
```

problem: when saving it as parquet, it gets splitted in four tables.
i believe this is caused by how spark processed the columns. maybe i could just join them in the final.

found a bug that was making the functions read only the first sentence of the content. the problem was in the carachters. better use json instead of csv.

now that i obtained a csv file with the preprocessed, i want to separate it based on categories so i can train an lda for each of them.

switching to `train_lda_02`

i created some scripts to split into categories.
i can now start making up the pipeline since i have all the code i need.
creating branch 'feature/pipeline-01'

WORKING ! !

Economie:

1. **Legea Bugetului:** Discuții despre adoptarea unei legi privind bugetul, cu accent pe scăderea dobânzilor și impactul asupra economiei.
2. **Execuția Bugetară:** Analize despre execuția bugetară, inclusiv deficitul și impactul asupra politicilor economice generale și sociale.
3. **Proiecte de Infrastructură:** Progrese și probleme legate de proiecte de infrastructură din România, cum ar fi pasaje și lucrări de supralărgire.
4. **Dezbaterea Bugetară:** Rapoarte și interpretări asupra deficitului bugetar, inclusiv controverse și declarații privind mediul economic.
5. **Educația Financiară:** Discuții despre necesitatea educației financiare, dezbateri despre reduceri sau majorări de prețuri în contextul economic actual.
6. **BNR și Politica Monetară:** Anunțuri și decizii ale Băncii Naționale a României privind politica monetară, cheltuielile și acoperirea financiară.
7. **Decizii Guvernamentale:** Hotărâri și decizii guvernamentale legate de sectorul bancar, hidrocarburi și alte aspecte economice.
8. **Politici Fiscale:** Dezbateri despre introducerea unui program de rable sau alte politici fiscale propuse sau adoptate.
9. **Negocieri și Joburi:** Discuții despre negocieri de muncă, poziții în cadrul Ministerului Finanțelor și altele legate de locurile de muncă.