Sprint 1 Concept

Introduction

Goal

Team 1

The goal is to make a simple python package which is able to do very basic brand sentiment tracking of a single article, without using any cloud technologies. The idea here is to get acquainted with the different elements to the brand sentiment tracking.

This way, we are not spending the first couple of weeks trying to learn all these tools without making any progress towards the project goal.

It would be good to also have an additional test script using `pytest` or `unittest` for each module. Don't worry about error handling or edge cases, just as long as it works given the right info.

Team 2

The goal is to learn how to run MLflow/Spark/Kubeflow locally and also how to deploy these tools on AWS. This idea here is that by the time we have a basic package working, we can start preparing the code for cloud deployment.

As a kind of deliverable, we could use one of the example projects (ideally written from scratch/tweaked and NLP-focused) and deploy it locally so everyone can see how the app will work and what the code looks like.

If it's not too much work (quite hard to say), we could also:

- Deploy it as a test app on AWS.
- Implement some GitHub workflows/AWS pipelines for pushing updates to the app.

Vision

After this sprint, the vision is to work on:

- Deploying to cloud, including:
    - Running the whole app locally using MLflow/Spark/Kubeflow tools.
    - Configuring AWS.
    - Setting up pipelines for continuous deployment.
- Improving different sections of the code.
    - In particular, how to make article extraction/parsing more efficient, by considering different possible implementations, e.g., whether we pull each article individually for processing or we do this in bulk.
    - Researching different NLP methods for identifying brands and sentiment and choosing the best model for our problem.
    - Researching different datasets for training our models.
- How to dump results into a database for our analytics dashboard to process.

Python Package

Repo Structure

This is a rough idea of what the python package could look like. My thinking is that we create a single repository on GitHub for it.

```
brand_sentiment/ # python package
        __init__.py
        extraction.py # for pulling articles from cc and preprocessing them.
        identification.py # for identifying the brand from some text.
        sentiment.py # for identifying sentiment from some text.

tests/
        extraction.py
        identification.py
        sentiment.py

brand_sentiment.py # main script for showcasing the package.
readme.md # shows what the package does and how to run it.
```

Package Init

This is quite basic but the idea is we can just pull the functions straight out of the package folder rather than having to look inside each module within the package.

```
from extraction import load_articles, preprocess_article

from identification import BrandIdentification
from sentiment import BrandSentiment
```

Article Extraction/Preprocessing

Goal

Create a python module which has two functions:

- One that, given a URL to an article, returns the content of the article as a string (e.g. `load_articles()`)
- Another that takes in an article and returns the visible text in a single string (e.g. `preprocess_article()`)

If all the articles have the same format, i.e. they're stored as JSON or YAML, it may be better to return that.

The code will mostly depend on how CommonCrawl store their articles in the S3 bucket. I'm not sure if it's possible to only get one article, or whether we get many. Also, they may be zipped up and may require some kind of authentication (though I doubt it).

Preprocessing articles could be quite tricky - I guess we picked a particular extractor that works well with most of the popular news sites.

Concept

```
import requests
import bs4

def load_articles(url: str) -> str:
        return requests.get(url).json()

def preprocess_article(document: str) -> str:
```

```
        parser = bs4.BeautifulSoup(document, type="html")

        return "\n".join(parser.findall("p"))
```

Brand Identification

Goal

Create a class that is able to return all of the brands identified in an article. I think it should probably take in all the text in an article and does any extra processing on article, e.g., splitting the article up by sentences.

Might also be worth returning the sentences associated with each brand so it's easier to do sentiment tracking on.

I'm not sure if it needs a neural network/training, I guess it depends on what methods are out there.

Concept

```
class BrandIdentification:

        def __init__(self):
                pass

        def predict(self, x: str) -> str:
                pass

        def train(self, x: List[str], y: List[str]):
                pass
```

Brand Sentiment

Goal

Create a class that is able to return the sentiment of either the whole article, or a select number of sentences (associated with each brand). This way we can loop through all of the brands and get the sentiment back for each.

Concept

```
from torch import nn

class BrandSentimentModel(nn.Model):
        def __init__(self, x):
                pass

        def forward(self, x: str) -> int:
                pass


class BrandSentiment:

        def __init__(self):
                self.model = BrandSentimentModel()
                ...

        def predict(self, x: str) -> int:
                pass
```

```
        def train(self, x: List[str], y: List[int]):
                pass
```

Main Script

```
import pickle

from brand_sentiment import *

test_url = "some url to an S3 bucket on CC"

with open('trained_sentimentiser', 'r') as fp:
        sentimentiser = pickle.load(fp)

with open('trained_identifier', 'r') as fp:
        identifier = pickle.load(fp)

document = load_article(test_url)
body = preprocess_article(document)

sentiment = sentimentiser.sense(body)
brands = identifier.identify(body)

print(f"Companies identified from URL are: {", ".join(brands)}")
print(f"Sentiment identified from URL is: {sentiment}")
```

Readme

Readme to go at the top of the repository that explains what the code does, how to run it, etc.

Could include these sections:

- Introduction
- Pre-requisites/Installation
- Usage