Technical Report - Cluster Reply Project

1. Identifying Topics

There are several possible approaches to get the most relevant topics. We will go through each one by one.

- There is the possibility to use existing datasets and taxonomies on social media. I tried looking for topics and wasn't very successful, however I found NewsCode, which is a metadata taxonomy for news topics (https://cv.iptc.org/newscodes/mediatopic/?lang=en-US); this list includes all possible news topics (a lot of them!) and it can be downloaded as a JSON object. There are of course way more than 20 to 100 topics in this dataset and one drawback would be that choosing a 20-100 subset of these topics would be difficult. **Suggestion:** Vectorise the posts and the topics, then compute the cosine similarity between each of these topics with the posts and identify the 100 most widespread topics.

- A more dynamic approach is to try to computationally derive the most relevant topics. This is possible through clustering the individual posts into 20 to 100 clusters as given by the problem statements. The centroid of each cluster will represent a single topic. However, since this procedure is completely unsupervised, we'll need to find a single topic for each cluster. This can be either be done by a human or by another model. **Suggestions:** 1) Using a generative model which outputs a single topic given a piece of text; 2) Given the list of topics in the previous approach, find the closest topic.

- There is a great product in the transformers library called BERTopic, which is a topic modelling (with a clustering approach) using BERT. It is available here => https://huggingface.co/blog/bertopic . It is quite easy to use and already trained on wikipedia, which is available here => https://huggingface.co/MaartenGr/BERTopic_Wikipedia

2. Application Architecture

When putting applications into production, there are certain factors we need to watch out for.

1. How much is the traffic going to be? What is the expected throughput?
2. Are there going to be peak usage times? How fast does the system need to scale?
3. Can we use our model on CPU during inference time? What is the accepted latency?
4. How much are we willing to pay monthly (for a cloud service)?

Managed services on the cloud can manage scalability to a great extent - however they might get costly.
I am going to assume that the system will get frequent usages, without having extreme peaks in usage.

Given these assumptions, we have certain candidates for hosting the model:

1. Azure App Service
2. Azure ML Studio
3. Azure Kubernetes Service
4. ...

In this case, I opted to deploy on Azure App Service. My reasons are:

1. App Service is a managed service (although all of the mentioned products are managed) and relatively easy to set up. However, it is almost the simplest solution in the given list above.
2. My second choice would have been Azure ML, simply because we will get much more MLOps capabilities for model monitoring. In the end I can say it is for the sake of simplicity that I have chosen App Service.

As for data and metadata, it is well-imaginable that the range of topics will change; we will have **data drift**. Therefore it's necessary to have regular re-trainings, either by implementing triggers or just in regular intervals.

My proposed final architecture will have the following elements:

1. **Storage:** The initial training data and the incoming inference data will be stored in Blob Storage. The incoming data will be cached for future retraining of the model, but data for the distant past will be deleted regularly. Here, I choose to cache 3 months worth of data. This comes as an architectural trade-off to facilitate scalability.

2. **Model:** The model will be deployed on Azure ML as an online endpoint. The REST API implemented via FastAPI serves the model.

3. **Pipelines:** Training Pipelines will be set up to retrieve data from storage, retrain and serve the model. These training pipelines will be invoked in regular intervals OR through data drift triggers.