ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

MSc in Business Analytics

Machine Learning and Content Analytics

"Smart Article Annotator & Recommender System"

Petros Tsotsi - f2822412

Michalis Athanasiou - f2822401

Thanasis Zygokostas - f2822402

Table of Contents

Part 1

1.1 - Introduction

The term “content” is used to describe anything created to convey information, ideas, or experiences to an audience with the use of multiple means of communication such as text, images, audio, video, graphics, etc. Humans are coming across various types of content every day in their lifetime, and it is of major importance this content to be processed appropriately so that they can deal with it effectively and efficiently. Towards this goal, the rapid expansion of AI and Machine Learning have resulted in the development of smart models that are able to understand content in any formats and perform multiple tasks that humans can significantly benefit from.

Let’s consider two major examples of AI models that process different types of content and we come across every day. A well-known example are the voice-controlled virtual assistants like Siri and Alexa. These systems are AI-powered helpers that you can talk to, ask questions, and give commands. Practically these systems are able to process audio content and help their users in various tasks. Another AI-based application that we come across everyday are the recommender systems used by all online platforms like Netflix, Spotify, Amazon and others. These systems suggest items to users based on their preferences, behavior, or similarity to other users or items, aiming in that way to promote to each user items based on their preferences, and achieve in that way higher revenues for the service providers.

These are only some typical examples of AI models thriving nowadays which, by being trained and by processing any type of content, are making peoples’ life easier. Inspired by these developments in field of AI and by the continuous challenges that humanity is facing, in this project we are aiming to build a smart system that will make use of content data to perform tasks that people and businesses can benefit from.

1.2 - Our project

In this project we are aiming to build an AI-based system that we call **Smart Article Annotator & Recommender System**, which implements two distinct but related tasks:

- the task of Article Annotation with semantic topics-tags
- the task of Article Recommendation based on their semantics

The first task that our system is going to implement is the Annotation – labelling of text content. More specifically, given the text of an article which is related to several topics, we aim to develop a

model that will be able to discover the semantics of the article and annotate it accordingly with the corresponding topics. Text Annotation, and Content Annotation in general, have multiple applications from using keywords and categories for items in online stores or news sites, to adding hashtags on content uploaded in social media. In this project we are going to approach this task as a multilabel classification problem where each given article text belongs to multiple topics and we have to predict those (among many others) and annotate the article with the predicted topics.

The second task of our project is the development of a Recommender system on text content. We aim to build a smart algorithm that given a text content, like an article, will be able to "understand" its semantics and suggest-recommend other articles of similar topic and semantics. To build this recommender system we will make use of pre-trained models (like BERTopic) that are able to "understand" the semantics of provided texts and group the texts of similar topics, and next we will apply clustering and similarity methods that will recommend semantically similar content to given articles.

1.3 - Our Vision/Goals

When setting the basis of this project and deciding what we are going to implement, our major goal was to develop a system/model/algorithm that would provide real value for the today's world. Value for both the users and the business side. After a lot of brainstorming, search, and discussions we decided that there is a lack of a smart AI-based system that can extract fast and accurately the semantics of any content that is present in content-rich platforms like social media, entertainment platforms, news platforms and others. A system that will be able to extract the topics of these contents automatically, without the need of human interference, annotate them accordingly, and also recommend similar contents to the users based on what they have already seen or uploaded.

The vision of this project is the development of a smart system that will support any content platform, such as social media, news platforms and others, and their users with two basic functionalities. On the one hand it will provide them with an automatic, fast and accurate topic detection and annotation of their text contents (articles etc.), and on the other hand it will provide a reliable and robust recommender system suggesting similar text contents to the users of these systems.

Let's consider the example of a user that is adding content on his/her social media account, for example publishing a new tweet or uploading a new photo. Wouldn't it be great for that user to be suggested with possible tags or hashtags to add to that content, that would reflect the topic and semantics of this new content? This practice would not only help the users label their content instead of manually adding tags, but it would also help the platforms improve their recommendation systems and have a much better view of the trends and the type of content that is added and trending.

Furthermore, after a user has published his/her desired content it would be great for the platforms to recommend to that user similar content that has been uploaded by other users. In that way every user will be immediately connected with contents and users with whom they share similar likings and preferences, which results in increasing users satisfaction and community growth.

In conclusion, our big vision is an AI-based system which can perform 2 distinct tasks. The first task is the annotation of any uploaded content with relevant labels that reflect the topic-semantics of this content, and the second one is the recommendation of semantically-similar contents for any content being uploaded. This system we are envisioning can be applied to almost every content-rich platform that users are uploading content. These are for example:

- Social media platforms like Twitter, Facebook, TikTok etc.
- News sites platforms like WSJ, NYT, Daily Mail, etc.
- Marketplace sites like skroutz, car.gr, etc.

At this point it is important to mention that, in terms of the present project and mainly due to resources limitation, we are going to focus exclusively on text content. We will proceed with the development of the above system implementing the goals of annotation and recommendation only on text content like a tweet, an article etc. Supporting also other types of content like images, video, audio will be considered in the Future Work section of Part 3.

Part 2

2.1 - Methodology

As also stated previously, our goal in this project is to build *a Smart Content Annotator & Recommender System* specialized for text content. A system that given a text will be able on the one hand to identify its topic and semantics and annotate it accordingly, and on the other hand to recommend similar text content to this given text. A system that we believe will have multiple applications on various sectors like news sites, social media etc.

To accomplish this 2-fold objective (Text Annotation & Text Recommendation) we had to search and find a dataset that will provide us with suitable data for training models that can be used for the annotation & recommendation tasks. After a proper dataset was identified we had to decide on how to approach our 2-fold objective problem. Our decision was to approach these two objectives as two separate problems and build relevant models for each case. Models that will be

trained on the same data but will act for these two different tasks (annotation & recommendation).

Starting with the **Text Annotation** task, we decided to approach it as a multilabel classification problem. More specifically, we consider the text content uploaded by any user as content that belongs to one or more categories, and what we are trying to achieve is to classify this content to these categories – labels. These labels will be then recommended to the user as the most suitable tags-hashtags to be added to his/her content before uploading. As in every multilabel classification task, we have to train a model on data having a wide variety of contents categorized in the corresponding categories. Then after sufficient training on that kind of data, the model will be able to handle new incoming content and classify it to one or more categories. Then the predicted categories will be proposed as candidate labels to the user for his/her uploaded contents. For the Text Annotation task we will proceed with developing several models starting from a simple sequential feed forward model, to a more complex RNN model (LSTM) and finally to fine-tuning a pre-trained model for sequence classification (DistilBERT).

On the other hand, for the **Text Recommendation** task and our goal of building a basic recommendation system for text content, we decided to make use of the Topic Modeling approach. *Topic modeling* is an unsupervised machine learning technique used to automatically identify latent topics within a collection of documents. It is particularly useful when dealing with large volumes of unstructured text, helping us understand the main themes and patterns without requiring manual labeling. Topic modeling has numerous applications from customer feedback analysis and trend detection to recommendation engines. In the context of this project, we are going to implement the Topic Modeling approach for Text Recommendation with the use of an advanced topic modelling method called BERTopic. **BERTopic** (Bidirectional Encoder Representations for Topic Modeling) [1] is a modern, transformer-based framework that leverages powerful language models like BERT to generate semantically rich document embeddings. It then applies dimensionality reduction (e.g., UMAP) and clustering (e.g., HDBSCAN) to group documents into topics. In terms of this project and our Text Recommendation task, we are going to experiment with the BERTopic model and apply different settings and setups that this model can support for the topic modelling of our data. Next, on top of the trained BERTopic that has captured the topics of our data, we developed and experiment with various recommendation methods that make use of the BERTopic captured topics, and perform recommendations based on the topics that documents are assigned to, and techniques like cosine similarity, inter-topic distances etc.

Finally after the methods/models for the two tasks are successfully developed, they are evaluated and used for inference to analyze and conclude about their capabilities and on whether they can satisfy our vision and goals that we posed in the Introduction.

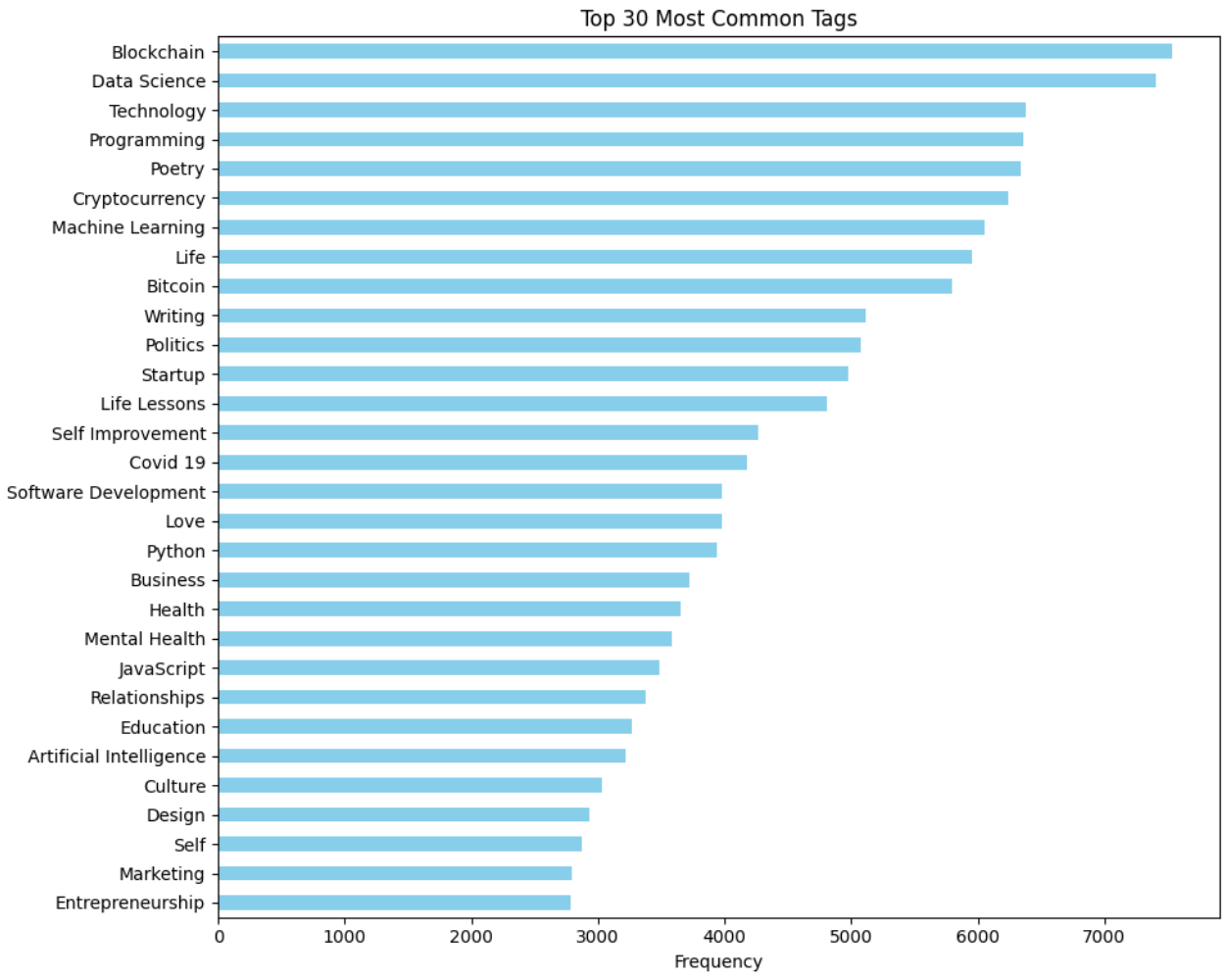
2.2 - Data Collection & Overview

A key challenge we faced after having set our goal for a smart system that will be able to annotate and recommend text data, was to find a suitable dataset for both of these tasks. After investigating multiple data sources (like Hugging Face, Kaggle etc.) and looking for data that comply with our objectives, we ended up to an Articles Dataset having all the features we were opting for.

The dataset is called "**190k+ Medium Articles**" and was found in Kaggle [2]. It comprises of more than 190 thousand articles that have been collected through scraping processes from the Medium website (https://medium.com/), a site where someone can find stories, articles, ideas of multiple topics and subjects. More specifically, the dataset includes **192.368 articles records**, scraped by the Medium website, that are available in a tabular format. For each of these records, which represent different articles published on Medium website, we find the following attributes:

- **title**: the title of the article
- **text**: the text content of the article
- **url**: the URL of the article on Medium website
- **authors**: the author(s) of the article (in list format)
- **timestamp**: the date & time that the article was published
- **tags**: the tags – labels associated to the article (in list format)

Figure 1: Most Frequent Articles Topics (Tags) in the dataset



To take a first look at our data we can visualize the most common article topics (tags) that are found in the articles of this dataset. As the previous Figure 1 bar chart illustrates the most popular articles are those referring to Tech topics like Blockchain, Data Science, Technology, etc.

2.3 - Data Processing/Annotation/Normalization

After this first overview of the Medium articles dataset that we are going to use for the training of our models, we continued with some preprocessing actions of the data before building and training our models. At first we kept only the text and tags attributes of the articles dataset since these are only needed for our tasks. For the annotation-multilabel classification task the text will act as the input, and the tags as the labels that the models should predict. Regarding the recommendation task, the text of the articles are sufficient. Therefore the rest of the attributes (title, url, authors, timestamp) were not needed and were removed.

Subsequently, instead of keeping the whole dataset which included a wide range of labels, many of which with very limited support, we decided to filter the data and keep only the most "popular" labels and the articles that belong to those labels. More specifically, we identified which are the most frequently appearing tags in our dataset, selected the top-30 of those tags as they can be seen in the previous Figure 1, and filtered the dataset to keep only the articles whose labels include at least one of the top-30 tags. After conducting this filtering we ended up with a dataset of **89.647 article records.** As the following bar plot in Figure 2 indicates, this updated dataset that we are going to work with has 55 thousand records - articles annotated with only one tag, and the rest 35 thousand having 2 or more topics.

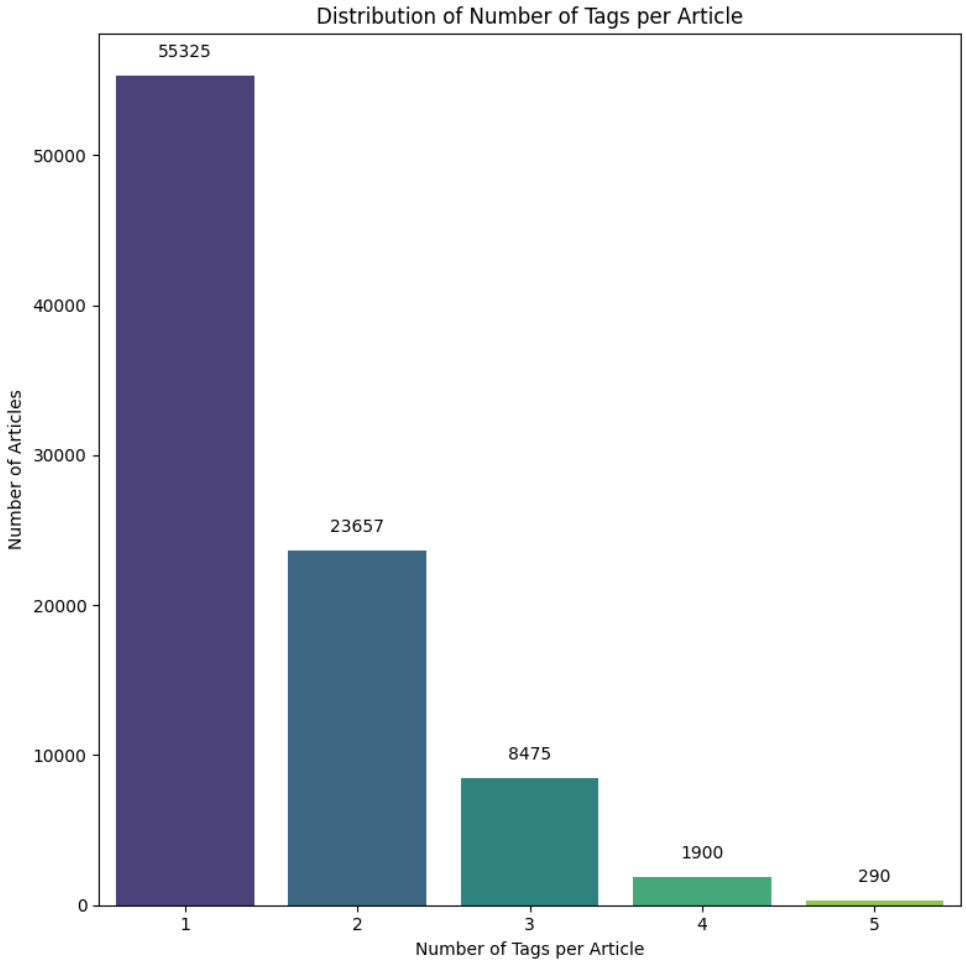


Figure 2: Distribution of the Number of Topics (Tags) per Article

Furthermore, to get a better understanding of the articles that we will be dealing with next, we proceeded with visualizing the Distribution of Articles Text Length in the following histogram (Figure 3). This plot makes it clear that we have to deal with texts which in general consist of a great number of words.

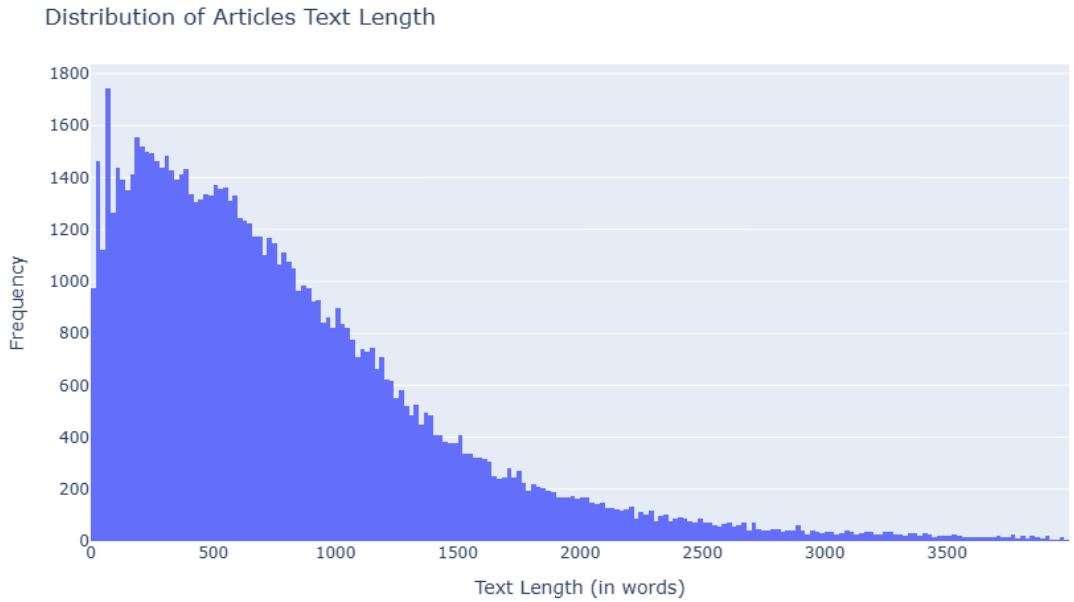


Figure 3: Distribution of Articles Text Length

Finally, ending the dataset overview analysis we also proceeded with a plot showing which topics tend to have the highest word length. From top-30 tags that we are focusing on and were shown in Figure 1, the bar plot of Figure 4 reveals that "Machine Learning" related articles have the highest number of words on average (more than 1.000 words) with Politics related articles following. Last come the Poetry articles with less than 200 words on average.

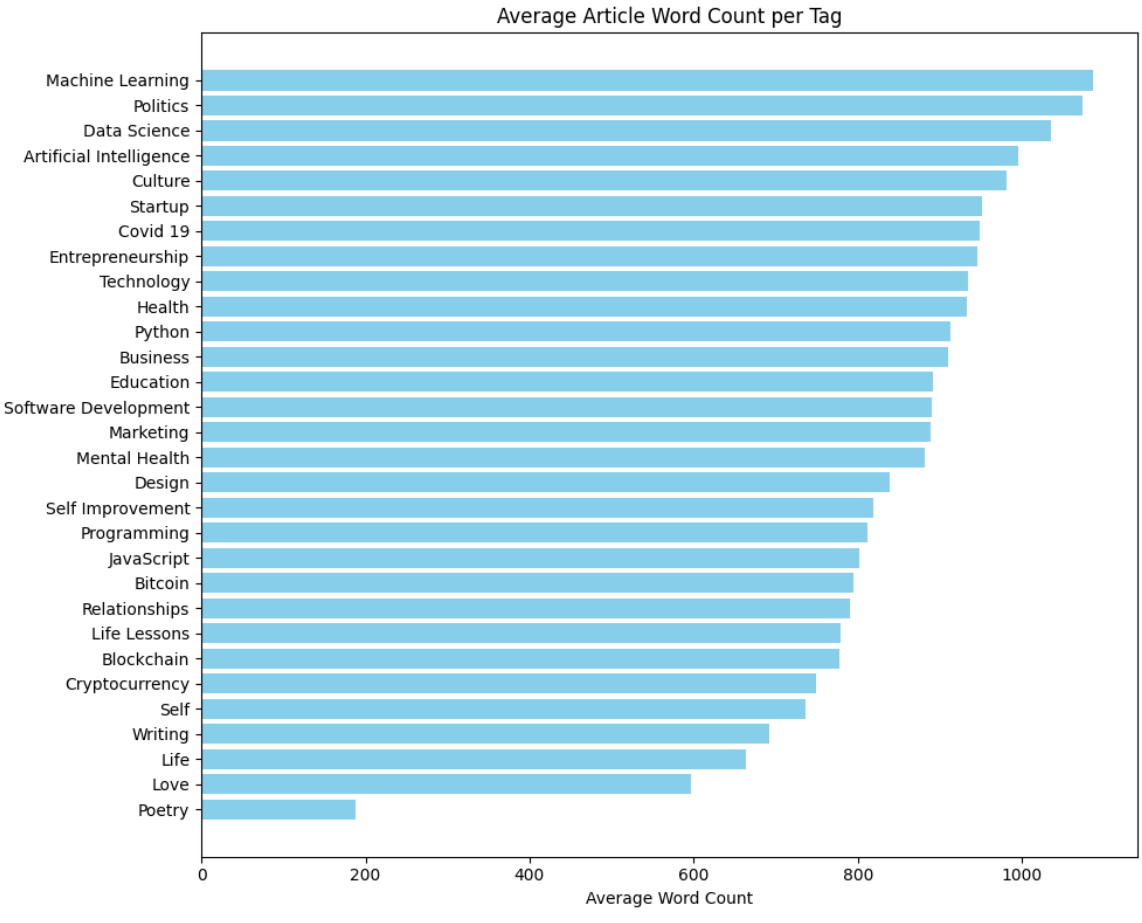


Figure 4: Average Article Word Length per Topic (Tag)

Another data processing task that it was necessary to perform before the models training (for the text annotation task only) was the splitting of the dataset into 3 separate subsets:

- ✓ The **train dataset** to be used for the training of our models
- ✓ The **validation dataset** to be used for the assessment of the models' performance during training
- ✓ The **test dataset** to be used exclusively for the assessment-evaluation of the models' performance after their training is completed.

To split our dataset into these 3 subsets we used the *train_test_split* method provided by scikit-learn which simply splits randomly a given dataset into subsets of specific size. In our case we selected to keep 15% for testing (13.448 records), and from the rest 85% to keep the 20% for the validation and the 80% for training. In conclusion, our splitting dataset procedure resulted in the following subsets:

- ✓ A **train dataset of 60.959 records**
- ✓ A **validation dataset of 15.240 records**
- ✓ A **test dataset of 13.448 records**

** At this point it is important to mention that we preferred to use this simple and random way of splitting instead of a Stratified Splitting because of the nature of the problem that we are tackling. Since we have multilabel classification problem, which means that each record can have one or more labels, stratification which will ensure equal distribution of the labels across all subsets is very difficult and challenging.*

2.4 - Algorithms, NLP architectures/systems

As stated also before, in this project we are going to cope with 2 different type of tasks: a Text Annotation task which will be approached as a multilabel classification problem, and a Text Recommendation task for which we will be using the BERTopic model for topic modeling on top of which several alternative recommendation algorithms will be applied.

Starting with the **Text Annotation task** that we consider as a multilabel classification task, we developed three distinct (3) models:

- ➢ A Sequential Feed Forward Network model
- ➢ An LSTM (Long Short-Term Memory) RNN model
- ➢ A fine-tuned DistilBERT model [3]

The **Sequential Feed Forward Network model** we developed is a simple feed forward network model designed to predict multiple labels for a given article text. It consists of an Embedding layer (that uses the pre-trained Glove embeddings of size 300) which is followed by a Flatten layer that concatenates the output embeddings into a unified vector. This output vector coming from the Flatten layer is then passed as input to two (2) consecutive Dense layers of 512 neurons each that use ReLu activation function, and a third Denser layer of 128 neurons using the ReLu activation function too. Each of these Dense layers is followed by a 0.4 Dropout layer. Finally the model ends with a Dense output layer of 30 neurons (corresponding to the topics - labels we have to predict) with sigmoid activation function since each article case can be matched to multiple labels.

The model's architecture with each layer parameters, output shape, and the total parameters (trainable & non-trainable) can be seen in the Table 1 that follows:

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 300, 300)	6,000,000
flatten (Flatten)	(None, 90000)	0
dense (Dense)	(None, 512)	46,080,512
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 512)	262,656
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 128)	65,664
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 30)	3,870

Total params: 52,412,702 (199.94 MB)
Trainable params: 46,412,702 (177.05 MB)
Non-trainable params: 6,000,000 (22.89 MB)
None

Table 1: Architecture Overview of the Sequential Feed Forward Network model

The **LSTM (Long Short-Term Memory) RNN model** is the second model that we developed, which in comparison to the previous one, is taking into account the sequence of words of the given input. Our LSTM model starts with an Embedding layer (that uses the pre-trained Glove embeddings of size 300) exactly as the previous Sequential model. The outputs of the Embedding layer are then passed into two (2) consecutive bidirectional LSTM layers (each of size 64). These bidirectional LSTM layers create representations of their given inputs by taking into account the context coming from words from both sides (left & right).

After these 2 layers, a final Dense output layer completes the model having 30 output neurons with sigmoid activation function as in our first model. A view of the model's architecture (layers, parameters, output shape) can be found in the next Table 2:

Model: "functional_1"

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 300)	0
embedding_1 (Embedding)	(None, 300, 300)	6,000,000
bidirectional (Bidirectional)	(None, 300, 128)	186,880
bidirectional_1 (Bidirectional)	(None, 128)	98,816
dense_4 (Dense)	(None, 30)	3,870

Total params: 6,289,566 (23.99 MB)
Trainable params: 289,566 (1.10 MB)
Non-trainable params: 6,000,000 (22.89 MB)
None

Table 2: Architecture Overview of the LSTM RNN model

The **fine-tuned DistilBERT model** [3] is the 3rd and final model that we implemented for the Text Annotation task. DistilBERT is a pre-trained Transformer [4] based model that can be used for various tasks including text classification. In our case we loaded the ***distilbert-base-uncased*** model from Hugging Face [5] (for the *multi_label_classification* problem and for *num_of_labels* = 30) along with its tokenizer. With the use of the DistilBERT tokenizer our train dataset was tokenized and made ready to be used for further training of the pre-trained DistilBERT. After the finetuning of the model is completed and without any change in its architecture, we use it for inference and evaluation on our test data.

Regarding the **Text Recommendation task** we firstly started with an attempt to "feed" the BERTopic model directly on our data without any preprocessing applied. This first exploratory attempt generated 896 topics. While it provided a broad map of the dataset, two problems quickly became apparent. Many topics were very small and fragmented, and some clusters were shaped by irrelevant or common words such as filler terms or repeated phrases. This made the interpretation difficult because the labels were noisy and often lacked a clear theme. To improve the quality of topics, we introduced a cleaning step. We created a custom cleaning function that removed URLs, punctuation, and numbers, converted all text to lowercase, and eliminated stop words using the NLTK library. We also added additional stop words that we identified as unhelpful in this specific dataset, such "like" or "one" words. Each article was cleaned and stored in a new column. This step ensured that the model focused only on meaningful words and was not distracted by noise, resulting in clearer, fewer, and more interpretable topics. After this preparatory

– introductory step , we proceeded with the implementation of 3 different BERTopic solutions (that we discuss next) on top of which the following 4 alternative recommender algorithms were implemented:

- **All-articles recommender**

All-articles-recommender algorithm works by calculating the similarity between the embedding of a chosen article and the embeddings of all other articles, using cosine similarity. It then returns the top closest matches, regardless of their topic label. This means that even if two articles are in different clusters, they can still be recommended together if they were semantically close. We also extend this recommender to unseen articles by cleaning the new input, generating an embedding, and comparing it to all stored embeddings. This allows us to recommend similar articles to completely new content.

- **Same-topic recommender**

Same-topic recommender algorithm, as its name reveals, is a recommender method that restricts itself to recommending articles only from the same topic that the given input articles is member of. It firstly identifies the cluster of the chosen article, and then calculates the cosine similarity only within that group. This alternative method produces highly focused recommendation results that stay tightly on theme/topic.

- **Hybrid recommender**

Another recommender alternative method we implemented to be used after the topic modeling by BERTopic is the Hybrid recommender. This approach is called hybrid because it combines semantic similarity between articles and topic alignment. For each candidate article the Hybrid recommender calculates both cosine similarity and a topic match score and then merges them into a hybrid recommendation score. Next for the recommendation process it uses a weighting that favors semantic meaning while still rewarding same-topic matches.

- **Topic-neighborhood recommender**

In this last recommender alterative that we call Topic-neighborhood recommender we focus on neighborhood recommendations. More specifically, instead of being restricted to the same topic cluster, this approach compares the article embeddings to the topic centroids, identifies the nearest topics, and returns candidates from neighboring clusters. This method provides more variety while still remaining thematically relevant.

In order to make use of these 4 recommender algorithms that we developed, we previously had to apply a topic modeling procedure which would cluster our text content articles into topic clusters. For this objective we implemented 3 different BERTopic solutions with each of them having different setup and settings selected before trained on our data. Our BERTopic alternative models that we experimented with are:

✓ BERTopic Method 1 (Default BERTopic)

In the first method, we applied BERTopic directly to the cleaned articles using the default settings. The model created embeddings for each text, reduced their dimensions with UMAP, and then clustered them with HDBSCAN. This resulted in a large number of topics once again, although cleaning reduced some of the noise seen in the very first exploratory run. However, there were still limitations, as the number of topics remained high and some were fragmented into small groups.

On top of this model we applied our 4 recommendations algorithms discussed previously. When trying the all-articles recommender and testing it on an unseen article about artificial intelligence and automation, it showed that the system returned highly relevant suggestions, many directly tied to artificial intelligence and autonomous vehicles. Next by applying our same-topic recommender and testing with an article about burnout and workplace stress, it returned only articles from the same burnout cluster, each highlighting different aspects of the same issue. This confirmed that the same-topic recommender was effective when narrow and consistent results were needed. Following, the hybrid recommender was tested with an article about neuroscience and the top results returned were all semantically close and also belonged to the same neuroscience cluster, balancing focus with meaning. Finally, with our topic-neighborhood recommender an entrepreneurship article brought back recommendations from both its own topic and closely related clusters about career transitions and founder challenges.

Overall, BERTopic Method 1 demonstrated that recommenders could work both for seen and unseen texts, and could flexibly provide narrow or broad results depending on the recommendation method approach. However, the large number of fragmented topics resulted in having limited interpretability.

✓ BERTopic Method 2 (Tuned BERTopic Parameters)

In the second BERTopic method for topic modeling it was aimed to improve the clarity and stability of the topics that the first approach lacked. We introduced a custom CountVectorizer that removed stop words, included both unigrams and bigrams, ignored very rare terms that appeared in fewer than five documents, and capped the vocabulary at 3000 words. This step ensured that topic keywords were more meaningful. We also tuned UMAP by setting the number of neighbors to 15, reducing to 5 dimensions, using cosine distance, and fixing the random state for reproducibility. These adjustments created tighter clusters that were easier for BERTopic to interpret. Finally, we required each topic to contain at least 20 documents, which reduced fragmentation by merging very small clusters into larger, more coherent ones. In addition, we enabled probability calculations for each article's topic assignment, which provided extra insight into model confidence. This configuration produced 382 topics, a significant reduction compared to earlier runs. The topics were larger, more coherent, and easier to interpret. The improvement in topic quality also carried over to the recommendation systems, which were applied in exactly the same way as in Method 1.

The all-articles recommender continued to perform well, but with clearer topic boundaries, the results became easier to interpret. For example, an article about the Covid-19 pandemic returned recommendations that covered mental health, brain chemistry, and statistics, all directly tied to pandemic themes, even when they came from slightly different clusters. The same-topic recommender also benefited from the clearer clusters. Testing with an article about burnout returned only articles from a strong burnout cluster, all discussing stress, remote work, and exhaustion in coherent ways. The hybrid recommender showed particularly strong performance under Method 2. Because the topics were stable and meaningful, combining semantic similarity with topic alignment produced very precise and context-rich recommendations. Even when new unseen articles could not be confidently assigned to a cluster, the recommender still produced semantically accurate matches, such as for texts about artificial intelligence and cloud computing. Finally, the topic-neighborhood recommender worked especially well in this method. With larger, well-separated clusters, identifying nearby topics added meaningful variety without introducing noise. For example, space exploration texts brought back articles from both space missions and astronomy-related clusters, enriching the recommendations.

Overall, BERTopic Method 2 produced the cleanest and most interpretable topics. All recommenders benefited from this structure, with the hybrid and neighborhood approaches standing out as the most effective.

✓ BERTopic Method 3 – MiniLM Embeddings

In our third and last topic modeling method using BERTopic model, we explored efficiency by using the MiniLM-L3-v2 embedding model from SentenceTransformers. Each cleaned article was converted into an embedding with this smaller model, then passed through UMAP and HDBSCAN as before. Unlike Method 2, we disabled probability calculations to save computation time. This configuration produced 667 topics, striking a middle ground between the fragmented structure of Method 1 and the consolidated structure of Method 2. Despite being a smaller model, MiniLM captured semantic meaning effectively.

The all-articles recommender returned strong results for both seen and unseen texts. For example, an article about Covid-19 retrieved others about mental health during the pandemic, while an unseen input about artificial intelligence suggested articles on AI and IoT, smart homes, and automation. The same-topic recommender worked best when the input belonged to a clear cluster, such as burnout or neuroscience. In these cases, it returned tightly focused results. However, when the input was labeled as an outlier, the recommender defaulted to pure semantic similarity, which was useful but less consistent than Method 2. The hybrid recommender again provided a balance between semantic similarity and topic alignment. For a neuroscience article, it retrieved related texts on neuroplasticity, brain-computer interfaces, and mathematical models of connectivity, all semantically close and within the same topic. Even for unseen texts that could not be confidently assigned to a cluster, the hybrid method still delivered relevant results, such as AI adoption and infrastructure articles. The topic-neighborhood recommender also proved effective.

It broadened recommendations beyond a single cluster while keeping them relevant, such as offering both startup challenges and career transitions in entrepreneurship contexts or covering both NASA missions and SpaceX projects for space exploration texts.

Overall, Method 3 provided a good compromise between detail and efficiency. While not as interpretable as Method 2, it was faster and still produced strong recommendations.

Part 3

3.1 - Experiments – Setup, Configuration

With the models being successfully defined and created for both Text Classification & Text Recommendation tasks, we proceeded to the next phase of our project which includes setting up all the necessary parameters and configurations for the models training, running the training procedures, and then evaluating their performance of the models on their relevant tasks. This phase is separate for the two tasks we are dealing with (Text Classification & Text Recommendation).

Starting with **Text Annotation – Text Classification task**, we have an already split dataset into 3 subsets (train, val, test) from the 2.3 section described previously. Now the next step before using these data is to bring them in a proper format to be used by our models (for training & inference). This step includes the encoding of the articles' text and the encoding of the articles' tags that our models need to predict for each article text.

For the **articles text encoding** we used the tokenizer provided by Keras with *num_words* = 20_000, which means that only the 20K most used words in the articles will be considered. Then this tokenizer was fitted ONLY in the articles of the train dataset and next was used to transform the article texts of all 3 subsets (train, validation, test). After that, a padding was applied to the article texts of all subsets to a max length of 300. As a result the shape of our article text data which are going to act as the input of our defined models are:

- Train dataset text shape: (60959, 300)
- Validation dataset text shape: (15240, 300)
- Test dataset text shape: (13448, 300)

As far as the **article tags encoding** is concerned, we used the MultiLabelBinarizer of Scikit-learn. After the MultiLabelBinarizer is firstly fitted on the train dataset tags, it is then used to transform the tag labels of all 3 subsets. In the end we ended up having for each record a binary vector of

length 30, with each position of that vector corresponding to each of the top-30 labels that each article corresponds to. The shapes of the dataset's labels is:

- Train dataset labels shape: (60959, 30)
- Validation dataset labels shape: (15240, 30)
- Test dataset labels shape: (13448, 30)

** The above encoded texts and tags are going to be used by all of our defined models, except of the DistilBERT for which we will use its own tokenizer for the articles texts encoding.*

Following the encoding of our data in the proper format for training, we continued with setting up some very important hyperparameters and settings regarding the models' training and performance evaluation. More specifically, for the Sequential and LSTM models we set the training to be done:

- ✓ for 30 epochs
- ✓ with batch size of 128
- ✓ with adam optimizer
- ✓ using as loss function the cross entropy loss
- ✓ using as performance metrics the AUC, Precision, Recall and F1 score
- ✓ using callbacks that include Early Stopping, Reduce Learning Rate and Model Checkpoint

Specifically for the pretrained DistilBERT model, the articles text are not fed to the model in the encoded form described above. Instead, the pretrained DistilBERT tokenizer is used to map the original articles text to the proper format that DistilBERT accepts (with 512 max input length). As for the fine-tuning of this model on our data we are going to use slightly different parameters due to resources limitation. More specifically it will be trained - finetuned:

- ✓ *for 3 epochs
- ✓ with batch size of 64
- ✓ using as loss function the cross entropy loss
- ✓ using as performance metrics the AUC, Precision, Recall and F1 score
- ✓ using callbacks that include Early Stopping, Reduce Learning Rate and Model Checkpoint

** (DistilBERT is a pretrained LLM with millions of parameters and its further training requires time and resources like GPU, RAM, etc that we did not have available. For this reason the fine tuning of this model will be done on less epochs and smaller batches).*

As far as the **Text Recommendation task** is concerned, we started with our initial dataset of the 89.647 article records that pertain to the 30 most popular topics. For this specific task we didn't need to apply any subseting into train or test subsets like in our classification task. Instead, we simply proceeded with feeding the data to the BERTopic models discussed previously in 2.4 section to achieve the topic modeling.

However, before training our BERTopic versions on the data we firstly introduced a data cleaning step. This step as a function on our data involved the removal of URLs, punctuation, and numbers from the article texts, converted all texts to lowercase, and eliminated stop words using the NLTK library. We also added additional stop words that we identified as unhelpful in this specific dataset, such as "like" or "one" words. Each article was cleaned and stored in a new column. This step ensured that the models would focus only on the meaningful words that are relevant to the texts topics and are not distracted by noise, resulting in clearer, fewer, and more interpretable topics.

After the data cleaning step is completed, the setup of our 3 BERTopic methods followed:

- The **BERTopic Method 1** uses the default BERTopic parameters. This includes the use of the *all-MiniLM-L6-v2* sentence-transformer model to generate sentence embeddings which help BERTopic group the article texts based on their meaning, and not just on word matching. In addition, we set the calculate_probabilities setting to False to make the process faster, especially since our dataset is big. By doing this, we saved a lot of time, but we also missed getting the probability scores that tell us how confidently each document belongs to a topic. All other BERTopic settings are the default ones.

- The **BERTopic Method 2** follows a completely different BERTopic setup. It makes use of a custom CountVectorizer that we defined which removes English stop words, uses only unigrams and bigrams, ignores very rare terms (that appear in <5 docs), and uses only the most frequent 3000 terms. It also uses a custom define UMAP function that is used for dimensionality reduction before the clustering is conducted, which compresses the text embeddings to 5 dimensions. Finally the BERTopic Method 2 has *min_topic_size* set to 20 which restricts the creation of tiny clusters.

- The **BERTopic Method 3** has a setup which appears more similar to that of Method 1. Same as in Method 1 it follows the default BERTopic setup with the exception of the embedding model. Method 3 uses a different embedding model from the default one that Method 1 makes use of (*all-MiniLM-L6-v2*). More specifically, it uses the *paraphrase-MiniLM-L3-v2* sentence transformer model which in comparison to the default one is much more cheaper and quicker to run especially for large texts.

3.2 - Results & Quantitative Analysis

The next step after defining the models for our two tasks and running the relevant experiments (training & evaluation) is to analyze their performance and results in the corresponding tasks. At first we will begin with the analysis of our models' performance on the Text Annotation – Text Classification task and next continue with the analysis of the results in the Text Recommendation task.

For the **Text Annotation – Text Classification task,** for which we created the three (3) models that were previously discussed, we firstly trained each one on the train dataset with the training parameters reported above in 3.1 section. After the training procedure was successfully completed, we used the trained models to predict the topics-labels-tags of each case included in the test dataset. Then, following the models' predictions we were able to analyze the results and their performance in the test dataset both in topic level (predictive ability of each model per label category), and overall across all labels using the relevant metrics like Precision, Recall and F1 score.

Focusing first on the scores achieved by the three (3) models we implemented and run on the Text Annotation task, we find the results presented in the following Table 3. In this table we can find for each model the average scores of Precision, Recall and F1 metrics in the test dataset. The average scores are used to summarize the models' performance across all labels-topics to be predicted. As it can be observed for each model and metric, both the micro and macro average score is reported. The macro average score reflects the average of the metrics across all classes regardless of their support, while the micro average counts the total true positives, false positives, and false negatives across all classes and is more resilient to class imbalances.

Model	Precision		Recall		F1	
	micro	macro	micro	macro	micro	macro
Sequential Model	66% / 41%		15% / 12%		25% / 17%	
LSTM Model	69% / 63%		43% / 39%		53% / 46%	
DistilBERT model	**72% / 67%**		**50% / 47%**		**59% / 54%**	

Table 3: Test scores of the 3 models on Precision, Recall, F1 score metrics (micro & macro average scores)

As the Table 3 scores reveal, **the best model across all metrics for the Text Annotation – Text Classification task is the finetuned DistilBERT model**. This is something we expected since this specific model in comparison to the other two is an already pretrained model, which means that it has an established understanding of text, and is also further trained (finetuned) on our train data for the specific task we are tackling. By checking also the rest 2 models scores we observed the LSTM model achieving far better scores than the Sequential model. This is an observation that we also expected. The LSTM model, in comparison to the Sequential, considers the order of the words/tokens of the text being processed which plays a key role in the text semantics and understanding. The Sequential, on the other hand, does not take into account the order but instead flattens the embeddings of the input text into a unified input which results in losing the words order. Furthermore, it is worth noting that the scores achieved by the LSTM model are pretty close to those of the finetuned DistilBERT, which by taking into account the time and resource demands of these 2 models (LSTM is far more faster and cheaper to train and run) we could conclude that the "secret" winner of this task is the LSTM model.

Specifically focusing on the "secret" winner model of this task – the LSTM model- we proceeded with visualizing its evolution of loss and metrics during the training process. In the following Figure 5 we can see the loss evolution both in the train and validation dataset as the epochs evolve. As we expected, the training loss is constantly decreasing as the epochs increase, while the validation loss decreases up to approximately epoch 10 and then it appears to be stabilized and slightly increasing as the epochs continue.

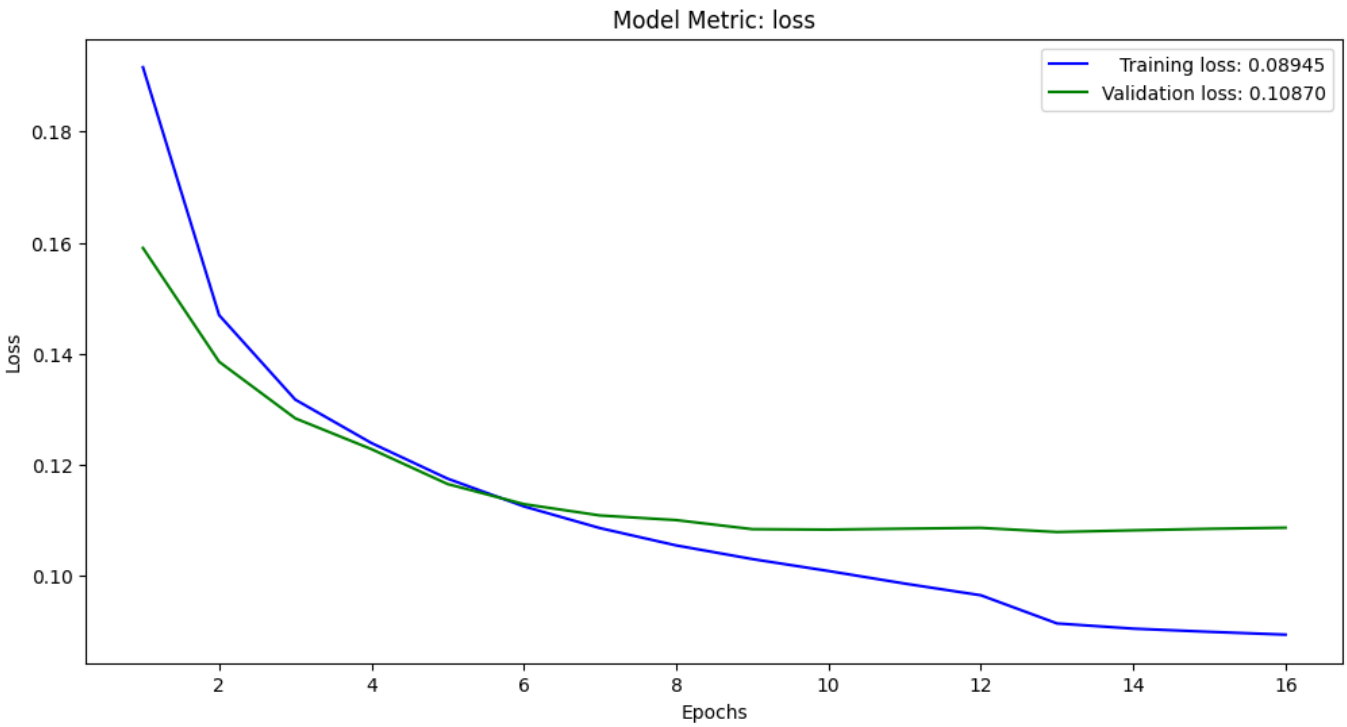
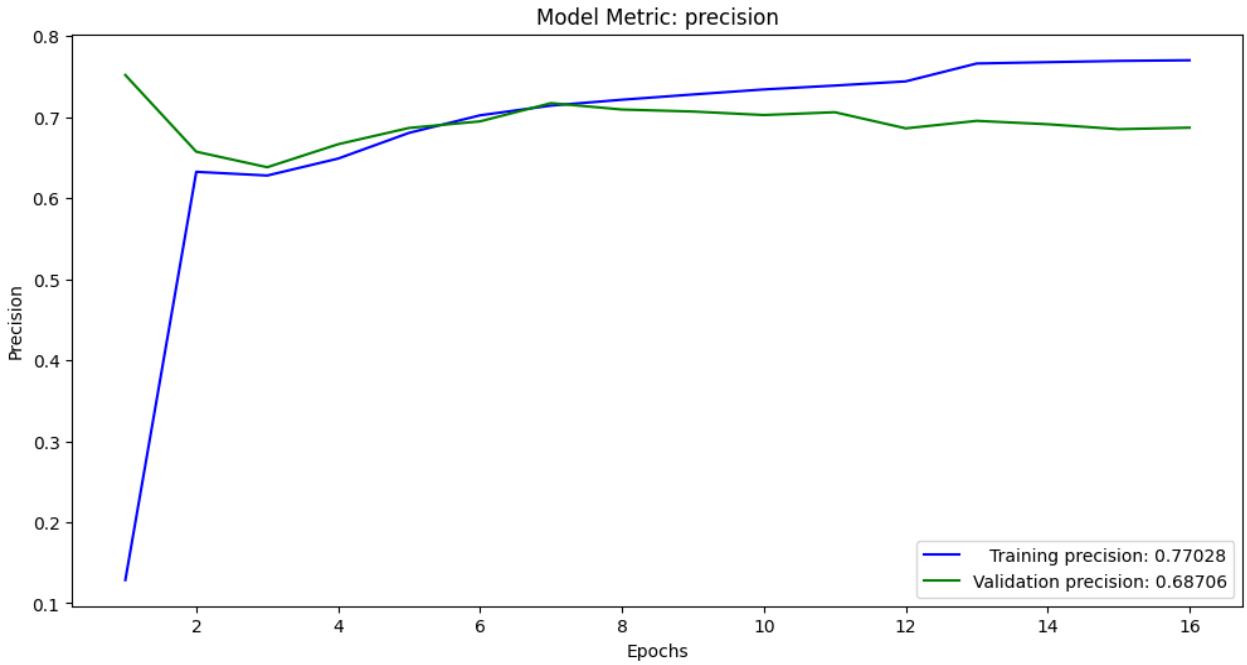


Figure 5: Train & Validation Loss evolution per epoch

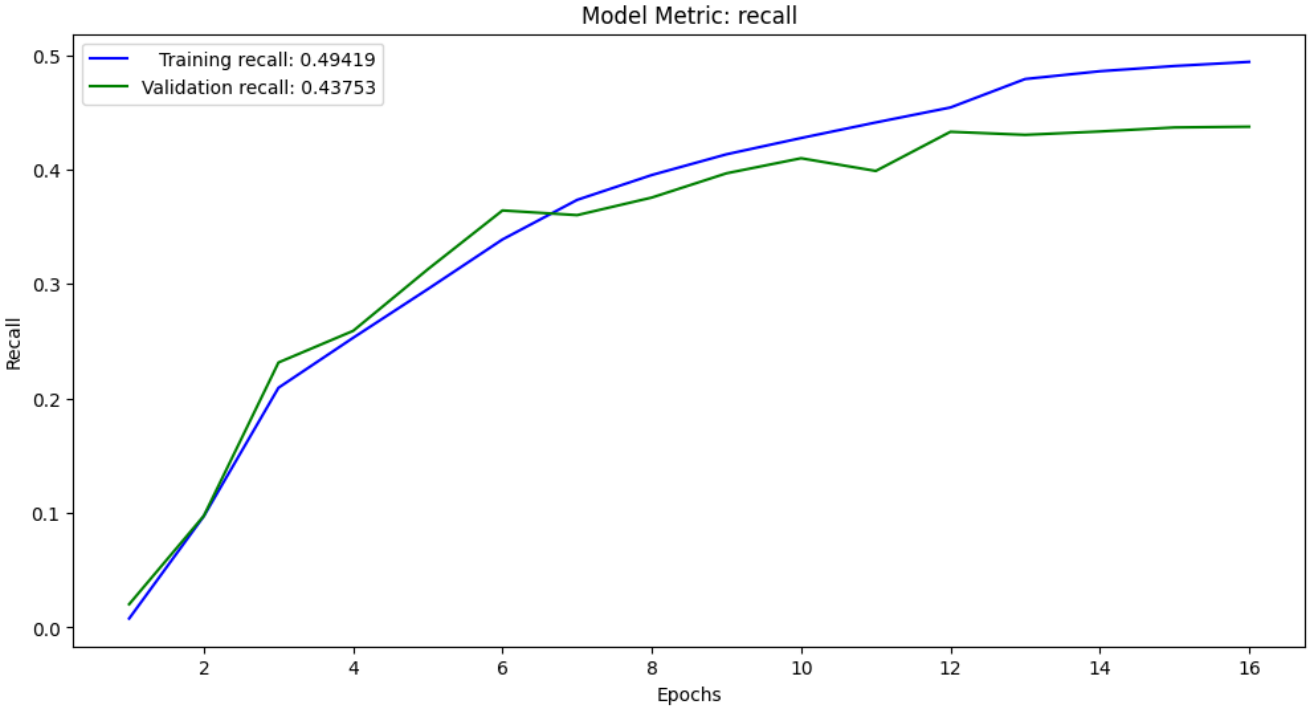
Next we proceeded with visualizing the Precision evolution as the epochs increased. By checking the relevant plot of the following Figure 6, we observe that after the 2nd epoch, where the precision stands around 60-70%, both the training and validation precision of the model is increasing. As the model is trained on more epochs, the precision appears to be slowly and steadily increasing reaching a peak at approximately 68% in the validation cases and around 77% on the train data.

Figure 6: Train & Validation Precision evolution per epoch



Similarly we visualized the Recall evolution of the model during the training. The view of the plot, shown in the following Figure 7, in this case appears to be quite different. As we can see, while the Recall started at zero scores in both subsets (train & validation), there was a slow but constant increase which after 16 epochs reached 49% on the train data and 43% on the validation dataset. These scores are significantly lower than the corresponding Precision scores, an observation that we are going to extensively analyze in the next section 3.3 – Qualitative & Error Analysis.

Figure 7: Train & Validation Recall evolution per epoch



Similar visuals are also produced for the other 2 models and can be found in the Appendix. For the Sequential Model: Appendix Figures A-1, A-2, A-3 show the Loss, Precision, and Recall evolution respectively. For the DistilBERT Model: Appendix Figures A-4, A-5, A-6 show the Loss, Precision, and Recall evolution respectively.

Regarding the metric scores that our three models achieved in the Text Annotation – Text Classification task for each specific label category, they are presented in the Appendix Tables A-7 (for the Sequential model), A-8 (for the LSTM model), and A-9 (for the finetuned DistilBERT model). An analysis of these results will follow in the next section 3.3 – Quantitative & Error Analysis.

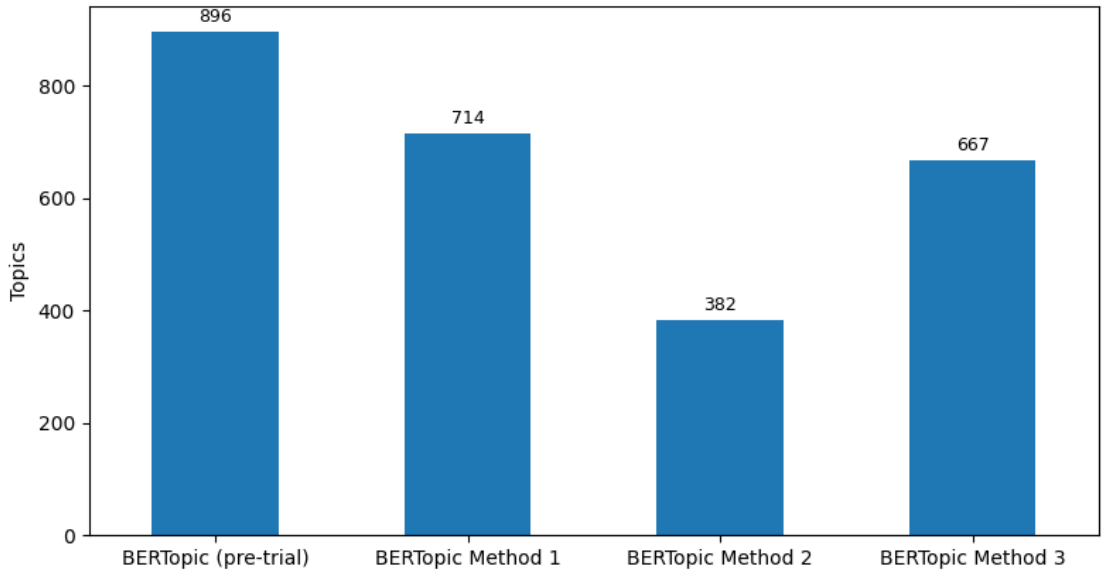
As far as the **Text Recommendation task** is concerned and as analyzed on the previous sections, our process of dealing with this task involved two subsequent phases:

- ➔ the first one which is about the topic modeling of our articles data using different BERTopic variations. The goal of this first phase is to identify dense clusters of topics inside our articles dataset.
- ➔ the second phase, which based on the resulting clusters that BERTopic has given us, applies one of our proposed recommender algorithms to recommend similar articles for a given text.

The first phase included the training of our 3 BERTopic methods (as discussed in 2.4 – Algorithms section). The result by training each of these BERTopic alternative models was to get topic clusters of our articles. However, two major aspects regarding the training of our BERTopic methods are the number of topics that each method produces, and the required time needed by each method to complete its training procedure. Regarding the topics that each method detected we can check the following bar plot shown in Figure 8. This diagram indicates significant variations on the number of resulting topic clusters in each BERTopic model, especially for BERTopic method 2 which has almost half the number of clusters that the rest of methods detected. This fact, however, doesn't necessarily mean that the quality of the detected topics will be lower than the rest of methods.

** BERTopic (pre-trial) method refers to an initial BERTopic run with the default parameters and without any text cleaning to have been applied*.

Figure 8: Number of topics/clusters detected by each BERTopic method



Another also important factor that needs to be taken into consideration is the amount of time needed for each method to complete its training procedure and finally capture the above topic clusters. This is what the following Figure 9 diagram is trying to visualize. More specifically, the following scatterplot shows the training time required by each method (in x-axis) vs the number of topics detected (in y-axis).

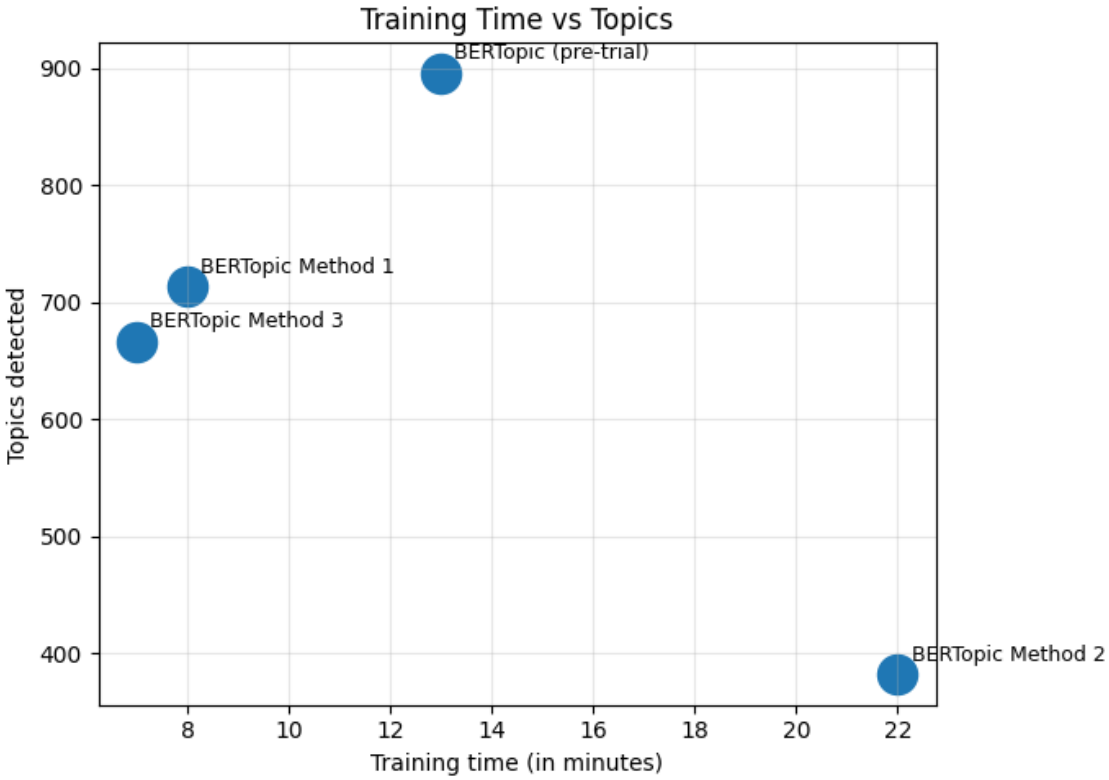


Figure 9: Number of detected topics vs Training time for each BERTopic method

As we can see, BERTopic methods 1 & 3 appear to have not only similar number of detected clusters, as it was also evident by the previous bar plot, but they also require similar time to get trained (approximately 8 minutes). On the other hand, BERTopic method 2 appears in the completely opposite side of the plot. It has significantly less clusters of topics detected, and requires much more time to be trained (22 minutes). These results indicate that BERTopic methods 1 & 3 appear to offer much similar features as far as the topic modeling phase is concerned, but their actual usability and differences will become more evident when on top of those we will apply our recommender algorithms (in the next section 3.3). On the other hand, BERTopic method 2 appears to have a different behavior with much higher time requirements and much less clusters detected. Whether this means lower quality recommendation results when the recommendation algorithms will be applied remains to be seen (in the next section).

After this discussion of our BERTopic methods results (on the number of detected topic clusters and the time needed to train), we will proceed in the next section 3.3 to apply the recommender algorithms on top of these BERTopic methods and identify the effect they have on the produced recommendations. Since no scores or statistics will be produced through this process, this analysis is conducted in the next section of the Qualitative Analysis.

3.3 - Qualitative & Error Analysis

Following the quantitative analysis of the results that we conducted in the previous section 3.2 for both tasks that we are tackling, the next step is a more deeper analysis of the results and the quality of the solutions that we are proposing.

Starting with the first task of **Text Annotation – Text Classification**, we presented the results of our three proposed models in Table 3 of the previous section. What we observed in this table is the significant difference between the Precision and Recall scores across all models. More specifically, while the models are achieving relatively high Precision scores on average, their average Recall scores appear at very low levels. As we can see, the finetuned DistilBERT model which has the best overall scores, has micro / macro Precision equal to 72% / 67% while its micro / macro Recall is just 50% / 47%. A similar view can be observed also for the rest two models.

This significant difference between the Precision and Recall scores across all our models in the Text Annotation task helps us make some useful conclusions. The high precision scores indicate that the labels the models choose to assign to a text article are with a large probability correct. On the other hand, the low Recall scores indicate that the models struggle to "extract" ALL the correct labels for an article text, and that there is a relatively high probability of missing some of them.

To better understand this behavior of our models in this task let's consider a simple example that is shown in the following Figure 10. In this example we have the first article text of the test dataset whose labels to be predicted are "Cryptocurrency", "Blockchain" and "Bitcoin" (*y_test[0]*). These labels in their encoded form appear as a vector of length 30 having ones (1) in the positions that correspond to these labels and zeros to all other positions. After using one of our models (in this case the LSTM model) to predict the labels of this text article, we can see that it predicted correctly one of those labels but missed the other two (see *y_test_preds[0]* vector).

```
[ ]  y_test[0]

     ['Cryptocurrency', 'Blockchain', 'Bitcoin']

[ ]  y_test_encoded[0]

     array([0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0])

[ ]  y_test_preds[0]

     array([0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0])
```

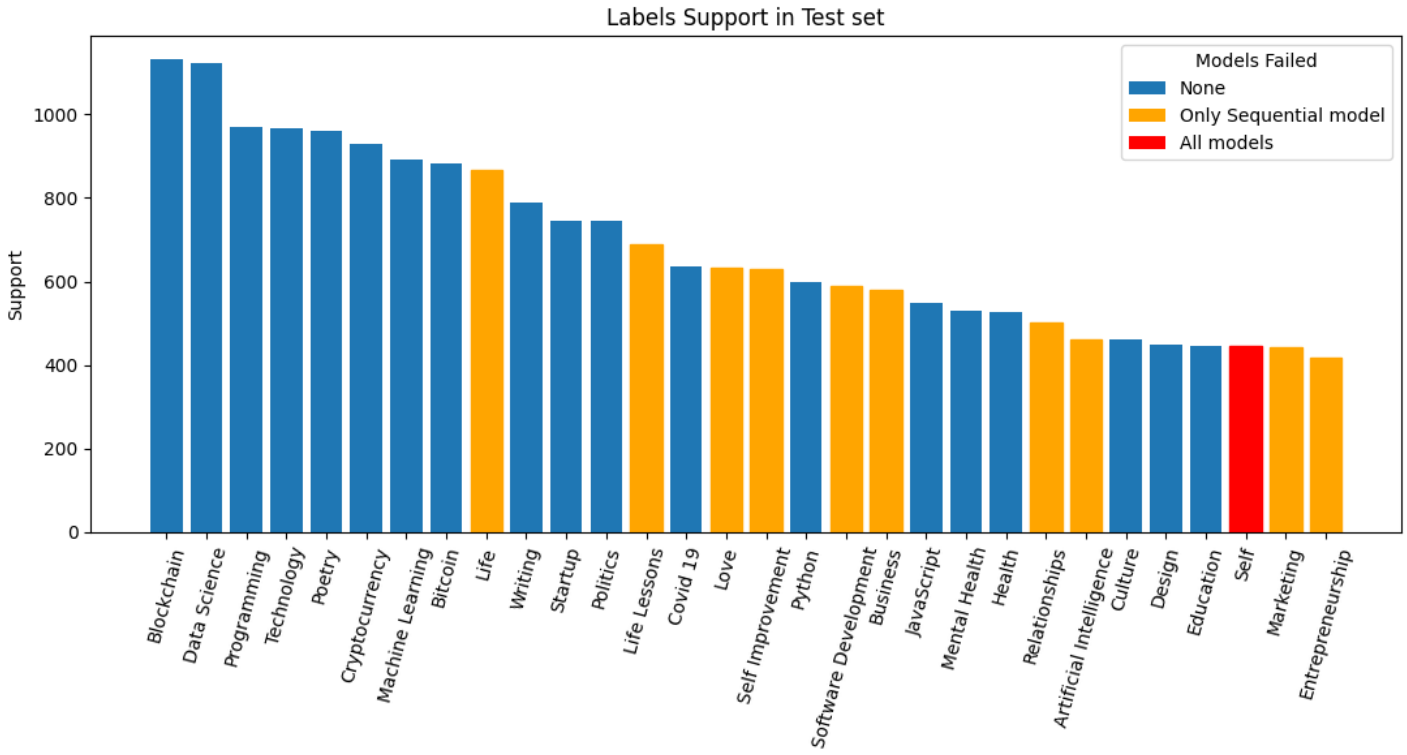
Figure 10: A Prediction example by the LSTM model

In terms of Precision and Recall, the model in this specific test case achieves 100% Precision since the label it predicted was indeed correct for that article, but its Recall is equal to 33% (1/3) since from the three correct labels the model managed to predict only one of them. This example describes accurately what the high Precision and low Recall of our models means when applied to specific cases of the task.

Another interesting aspect to discuss are the scores that our models achieved in the Text Annotation task **per label-category**. By looking at these scores presented in separate tables for each model in the Appendix *(Table A-7 for the Sequential model, Table A-8 for the LSTM model, Table A-9 for the finetuned DistilBERT model)*, we can observe that there are some specific label categories in which our models have zero (0.00) score in all three metrics Precision, Recall, F1. More specifically we can see that the Sequential model, which was expectedly by far the worst among our models, has zero score in the following 11 out of 30 label-categories: *Artificial Intelligence, Business, Entrepreneurship, Life, Life Lessons, Love, Marketing, Relationships, Self, Self Improvement, Software Development*. On the other hand, the LSTM & the finetuned DistilBERT models had zero score only in the "Self" label category. One possible interpretation of these zero scores for some label-categories can be their poor support in the test dataset on which the models are evaluated.

We can better understand this by having a look at the bar chart displayed in the following Figure 11. This bar chart visualizes the support of the tag-labels in the test dataset, with the bar corresponding to each label being colored based on which models fail to achieve a non-zero score on that label. More specifically, the bar chart indicates that the only tag-label in which all models are having a zero score (colored in red), is the "Self" category, that has the 3rd lowest support in the test dataset. Furthermore, the bar chart shows the rest of labels in which only the Sequential model is failing in orange color. These labels also are observed to have in general low support in the test dataset.

Figure 11: Labels Support in the Test dataset colored



On the **Text Recommendation** task our analysis focus is on the three BERTopic models we implemented along with recommendation algorithms we developed to run on top of these topic modeling alternatives. When comparing results across the three BERTopic methods, we can see important differences in how each recommender behaved, both for articles that were already in the dataset and for brand-new unseen articles.

Starting with the **all-articles recommender**, which is based purely on semantic similarity, the outcomes were consistent across all methods but not equally easy to interpret. In Method 1, where the model created 714 topics, the recommender was able to bring together articles that were semantically close, such as when a new text about artificial intelligence and automation was tested and the system returned pieces about autonomous vehicles and digital transformation. While thematically relevant, the large number of fragmented clusters made it harder to understand how the articles related in terms of topics. In Method 2, where we tuned the model and reduced the number of topics to 382, the recommender's results became much clearer. For example, an article about Covid-19 returned related pieces on mental health, brain chemistry, and statistics, all clearly linked to pandemic discussions. Because the topics were larger and more stable, the results were easier to explain and more trustworthy. In Method 3, using MiniLM embeddings, the recommender again produced strong matches, such as grouping together texts about AI and IoT or Covid-19 and mental health. The results were similar in quality to Method 2, although interpretability was slightly lower because the topics were more numerous than in Method 2. For unseen articles, the same pattern held: Method 1 produced relevant but scattered matches, Method 2 returned highly precise sets, and Method 3 balanced efficiency with solid thematic coverage.

The **same-topic recommender** showed the biggest contrast across methods. In Method 1, because many clusters were very small and fragmented, the recommender only gave a very narrow set of results, sometimes repeating very similar perspectives, instead of offering a broader but still relevant range of articles. For example, an article about burnout only returned other burnout-related texts, which was useful but reflected the fragmentation of the topic space. In Method 2, the improvements in topic quality made a clear difference. When we tested with the same burnout article, the recommender returned five articles, all in the same cluster, covering stress, remote work, and exhaustion in ways that were coherent and easy to interpret. This showed the strength of larger, well-defined topics. In Method 3, the same-topic recommender again worked well when the article belonged to a strong cluster, such as burnout or neuroscience. However, when an article was assigned to the outlier category, the recommender had to fall back on semantic similarity alone. This meant it was less stable than Method 2 but still useful. For unseen articles, the recommender in Method 1 often struggled because outlier assignments were common, while in Method 2 it performed better thanks to stronger topic structures, and in Method 3 it provided reasonable results but with less consistency than Method 2.

The **hybrid recommender**, which combines semantic similarity with topic alignment, was the most consistently strong across all three methods. In Method 1, it returned results that were relevant and semantically close, such as for neuroscience articles where the recommendations were tightly focused. In Method 2, this recommender gave its best performance, because the topics were

stable and interpretable, combining semantic similarity with topic alignment produced highly accurate results. For example, an article on neuroscience brought back related texts on neuroplasticity, brain-computer interfaces, and brain connectivity, all within the same topic and semantically close. For unseen texts, such as an article about artificial intelligence and cloud computing, the recommender was able to provide coherent results even when the model could not confidently assign the new text to a topic. This showed its robustness in real-world scenarios. In Method 3, the hybrid recommender also performed well. For example, an unseen text about AI again produced highly relevant results on AI adoption and IoT, even when the text was labeled as an outlier. The main difference was that in Method 2 the hybrid recommender worked with clearer topic structures, making the results easier to interpret, while in Method 3 it leaned more heavily on semantic similarity when topic assignments were uncertain.

As for the **topic-neighborhood recommender**, it showed how different the methods could be when it came to diversity in recommendations. In Method 1, because the topics were so fragmented, adding neighboring clusters often introduced noise. For example, articles about entrepreneurship would bring in loosely connected results. In Method 2, this recommender became much more effective. An article on entrepreneurship returned a balanced set that included startup challenges, legal preparation, and career transitions, all from related clusters that made sense together. This produced both focus and variety. For unseen texts, such as one about space exploration, the neighborhood recommender in Method 2 returned articles about NASA missions, SpaceX projects, and astronomy research, showing that it could capture both direct and related perspectives. In Method 3, the recommender also performed well, especially for unseen articles. For example, a text about space exploration returned results about multiverses, wormholes, and commercial space programs, while a political text about Donald Trump returned a tightly focused set of articles all about U.S. elections.

In summary, the first method was useful as a starting point, but the topics it produced were too fragmented, which made them harder to understand. The second method gave the clearest and most reliable results across all recommenders, so it was the strongest overall. The third method was faster and still produced good results, but it sometimes had to rely more on similarity in meaning when the topics were not very stable. Across all methods, the hybrid recommender was the most reliable, especially for new, unseen articles. The topic-neighborhood recommender worked best when the topics were strong, as it gave a good mix of focus and variety.

Now with the BERTopic Method 2 as a starting point for topic modeling of our articles dataset, we are going to proceed with applying our 4 recommender methods on real article texts aiming to show in practice their recommendation results on real text content that has been found online. At first we are going to recommend on an article text related with a topic that BERTopic has seen, for example, Politics. Next, we are going to test our recommenders on an article of completely irrelevant topic in comparison to what BERTopic has been trained on (e.g. football).

Elon Musk says he is launching a new political party, weeks after dramatically falling out with US President Donald Trump. The billionaire announced on his social media platform X that he had set up the America Party, billing it as a challenge to the Republican and Democratic two-party system […]

Table 4: Recommender algorithm results on the given Politics-related article

Recommender algorithm	1st Recommendation	2nd Recommendation	3rd Recommendation
All-articles recommender	Will Donald Trump Run Again In 2024? Yes. Donald Trump is a President like no other. Political pundits and news organizations have counted him out since 2015 when he first announced he would run. President Trump has not followed the unspoken rules of being the president, …	Republicans Will Continue Embrace of Trump Donald Trump's tumultuous time as president will be coming to an end in a little over six weeks time. Calls for the Republican Party to distance itself from the problematic figure have been abundant as moderates try to wrestle back…	Elon Musk 5 Rules for Startup Founders and Entrepreneurs for Success in 2019 When every expert in the car industry said that he would fail with Tesla, Elon Musk took to sleeping on the factory floor. What's more, he has already defied the law of gravity …
Same-topic recommender - Hybrid recommender - Topic-neighborhood recommender	Will Donald Trump Run Again In 2024? Yes. Donald Trump is a President like no other. Political pundits and news organizations have counted him out since 2015 when he first announced he would run. President Trump has not followed the unspoken rules of being the president, …	Republicans Will Continue Embrace of Trump Donald Trump's tumultuous time as president will be coming to an end in a little over six weeks time. Calls for the Republican Party to distance itself from the problematic figure have been abundant as moderates try to wrestle back…	Trump Is Losing Twitter Followers for the First Time in 4 Years Many don't remember that in October of 2016, when Trump thought he'd lose the election, he was ranting about his certainty of voter fraud. In all of his rallies up until election day, he claimed, …

In this test case we have an article text related with Politics and referring to Elon Musk willingness to create a new political party after breaking up with Donald Trump. The Politics topic is quite usual in the dataset that our BERTopic was trained on, so we expect our recommenders to easily recommend relevant articles. As the above results table indicates, our 4 recommender algorithms gave pretty similar recommendations. More specifically, by focusing explicitly on the top-3 recommendations of each method, we observe that with the exception of the all-articles recommender, all other recommenders gave exactly the same articles as recommendations, while the all-articles recommender differed only in one.

In fact all 4 methods are proposing two articles relevant with Donald Trump and his engagement with elections and his party but none of them is relevant with Elon Musk. The difference is made in the 3rd recommendation where the all-articles recommendation, which calculates the cosine

similarity between all articles regardless their topic, gives an article relevant with Elon Musk. On the other hand, the rest 3 methods are recommending another article relevant with Donald Trump. These results indicate that all-articles recommender, that is not restricted inside the topic of the input article, is able to recommend a broader set of content while at the same time remaining in topic.

UEFA have confirmed that Chelsea's Champions League squad has hit the required financial barometers after extra restrictions were placed on the club earlier this year. Chelsea had been fined more than £27million for breaching regulations last season whilst in the Conference League. Had Chelsea not complied with European football governing body's new measures then their fine […]

Table 5: Recommender algorithm results on the given Football-related article

1st Recommendation	2nd Recommendation	3rd Recommendation
The Goalkeeper's Anxiety at the Penalty Kick (And Other Anxieties) It's stoppage time at Vicarage Road. Watford goalkeeper Ben Foster has been given the go-ahead to attack a free kick inside the Chelsea penalty area. Deulofeu's delivery finds Doucoure, who flicks on a header which is attacked by Foster, …	Liverpool FC Being Boss in Real life is Ruining FIFA 21 for Me, a Liverpool Fan To be fair, while a lot of Liverpool fans would describe the last 30 years as a 'dark period', it's not been without highlight. In fact, I think most other football clubs would kill to have dark periods even half as successful as ours. …	Premier League Power Rankings: Matchweek 14 The schedule already being congested due to Covid-19 and the start of the festive fixtures means the next several weeks in the Premier League are coming faster than we expect. …

Now moving on to a more challenging case of recommendation, we suppose to have a user uploading an article relating to sports and more specifically relevant with football and Chelsea football club. This is a challenging case since our BERTopic model was exposed on quite a few cases of this specific topic. What we can observe from our recommenders results is that all of them recommended the same top 3 articles in response to the given article. More specifically, the 1st recommended article is about a specific event that happened in a game of Chelsea which is what the given article was talking about. The following 2 recommended articles were also related with football but not strictly related with Chelsea. Instead, they are speaking about other English clubs or the Premier league in which Chelsea participates. This priority of recommendations also indicates that our recommenders are able to recognize and quantify (for example through cosine similarity) how much similar are the articles and recommend the articles in order that is relevant with their similarity.

3.4 - Discussion, Comments/Notes and Future Work

Our main attempt throughout this project was to develop a smart system that will be able to perform two main operations on content being uploaded on any kind of platform like social media, news sites, and others: the first was to automatically annotate any uploaded content with labels that reflect the semantics of this content, and the second one was to suggest-recommend similar content to the uploaded one.

Our approach was to deal with these two operations separately. On the one hand, we developed three candidate models for the Text Annotation – Text Classification task. Among these models we found the finetuned DistilBERT model to achieve the best results, with the LSTM model following with slightly worse performance but much cheaper in time and resource requirements.

On the other hand, to achieve an effective Text Recommendation we made use of 3 different BERTopic alternative architectures for topic modeling on top of which 4 different recommendation alternative algorithms were proposed. The three topic modeling methods based on BERTopic model showed different trade-offs. Method 1 proved that BERTopic could map the dataset, but the topics were noisy and fragmented. Method 2 gave the best performance, with clear and stable topics that made all the recommenders work much better. Method 3 provided a good middle ground, producing solid results while running faster, which makes it useful when efficiency is important. Regarding our proposed recommenders, the hybrid one was the most effective, especially for unseen articles. The topic-neighborhood recommender was also useful when we wanted more variety in results. Overall, Method 2 combined with the hybrid recommender gave the best balance of clarity, accuracy, and reliability, while Method 3 is a good choice for larger or faster applications.

What is important to be underlined is that in terms of this project we focused on achieving our 2-fold objective (annotation & recommendation) only for text content. This was due to multiple limitations we had that didn't allow us support also other types of content. These were the limited resources we had access to in order to run our experiments, and the lack of reliable and appropriate datasets to train models on processing other types of content too. As a future work, we consider expanding our proposed *Smart Article Annotator & Recommender System* to other types of content data such as images, videos, audio which are also a major part of the content being uploaded on platforms like the social media, marketplaces, news platforms etc.

Another point of discussion, that could be an interesting idea for a future work, would be to proceed with a more advanced solution that would combine the two tasks we dealt with in this project and not just apply them separately. For example one idea would be the Text Recommendation process to utilize the predicted topics-labels of the Text Annotation task in making even better recommendations. Practically the Text Annotation task models would firstly run to predict the labels of a specific article text, and next after that by searching which clusters have these labels-topics as their characteristic ones, the Recommendation models would use these predicted topics on searching the best article suggestions.

The combination of these two tasks could also be applied inversely. More specifically, we could firstly make use of the Recommendation models to identify the top suggested articles for a specific article. Then to identify the topic(s) of this specific article of interest, we could apply our Text Annotation models to the suggested articles, apart from the article of interest, so in that way we can more precisely predict the topic semantics of our article.

Finally, a consideration for future work on this project is the development of a mockup application that would demonstrate directly and interactively the functionalities we aim to support. This application could be a simple web page which would allow any user upload any kind of content (in our case we support only text) similar as they would do in a social media platform. This web page, before the final upload of the users' content, would run the best model we implemented for the Text Annotation task to predict the topics-labels that fit to that uploaded content and would suggest the user to annotate his/her content with these predicted labels. After that, the user would be suggested-recommended with other uploaded content by other users that is similar to what he/she uploaded. To achieve this, the application would run our proposed recommendation engine based on the content topics and the identified clusters of topics.

Part 4

4.1 - Members/Roles

Our team consists of 3 members, all full-time students of the MSc in Business Analytics at AUEB for the academic period 2024-2025. The members of the team come from diverse backgrounds that we believe was a key factor in achieving this high level work. More specifically our team's members are:

Michalis Athanasiou who has a Bachelor's degree in Business Administration with a specialization in Information Systems from Athens University of Economics and Business. Michalis after completing his undergraduate studies, worked as a salesman for Apple, as a project manager for a consulting company, and then attended the MSc in Business Analytics at Athens University of Economics and Business as a full-time student. Michalis' role in the project included the definition of the business goals that our system is set to achieve. Furthermore, Michalis took over the investigation and finding of a suitable dataset that met our goals and objectives, and also actively participated in the development of the Recommender algorithms of our solution.

Petros Tsotsi who has a Bachelor's degree in Informatics from Athens University of Economics and Business. Petros after completing his undergraduate studies, worked for 2 years in a well-known software company as Business Analyst and Quality Assurance Engineer, and then attended the MSc in Business Analytics as a full-time student. Petros' role in the project included the development of the models for the Text Annotation - Multilabel Classification task as well as analyzing their performance and results.

Athanasios Zygokostas who has a Bachelor's degree in Mathematics with a specialization in Applied Mathematics from the National and Kapodistrian University of Athens. Athanasios after completing his undergraduate studies, worked in various roles including seasonal logistics projects, hospitality, and private tutoring in Mathematics, and then attended the MSc in Business Analytics at Athens University of Economics and Business as a full-time student. Thanasis' role in the project included the analysis and preprocessing of our dataset, and the design and implementation of the Recommender solution of our system (BERTopic methods, recommender algorithms).

4.2 - Time Plan

The implementation of the current project involved significant effort from all team members that were previously presented. It included the execution of multiple steps from identifying the business needs that are going to be met, finding the appropriate data for our goals, and up to finally running our models and presenting their performance. The time plan indicating all the steps needed to complete this project along with an approximate timeframe on which each step took place is presented in the following Table 6.

Activity – Process	***Lasted (From – To)*** ** approximately*
Identify Project Idea & Business Goals	15th May – 1st June
Search & Find Data for our Goals	1st June – 10th June
Data Analysis & Preprocessing	10th June – 20th June
Models Design, Development	20th June – 30th June
Models Training & Performance Evaluation	1st July – 31st July
Models Results & Performance Analysis	1st August – 15th August
Project Report & Presentation	15th August – 9th September

Table 6: Time Plan of the Project's Activities with lasting period

4.3 - Bibliography

1. *BERTopic: Neural topic modeling with a class-based TF-IDF procedure,* https://arxiv.org/abs/2203.05794

2. *190k+ Medium Articles by Fabio Chiusano,* https://www.kaggle.com/datasets/fabiochiusano/medium-articles/

3. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter by Sanh et al.,* https://arxiv.org/abs/1910.01108

4. *What is a Transformer Model by Nvidia Blogs,* https://blogs.nvidia.com/blog/what-is-a-transformer-model/

5. *DistilBERT model by Hugging Face,* https://huggingface.co/distilbert/distilbert-base-uncased

Appendices

Figure A-1: Train & Validation Loss evolution per epoch of the Sequential model

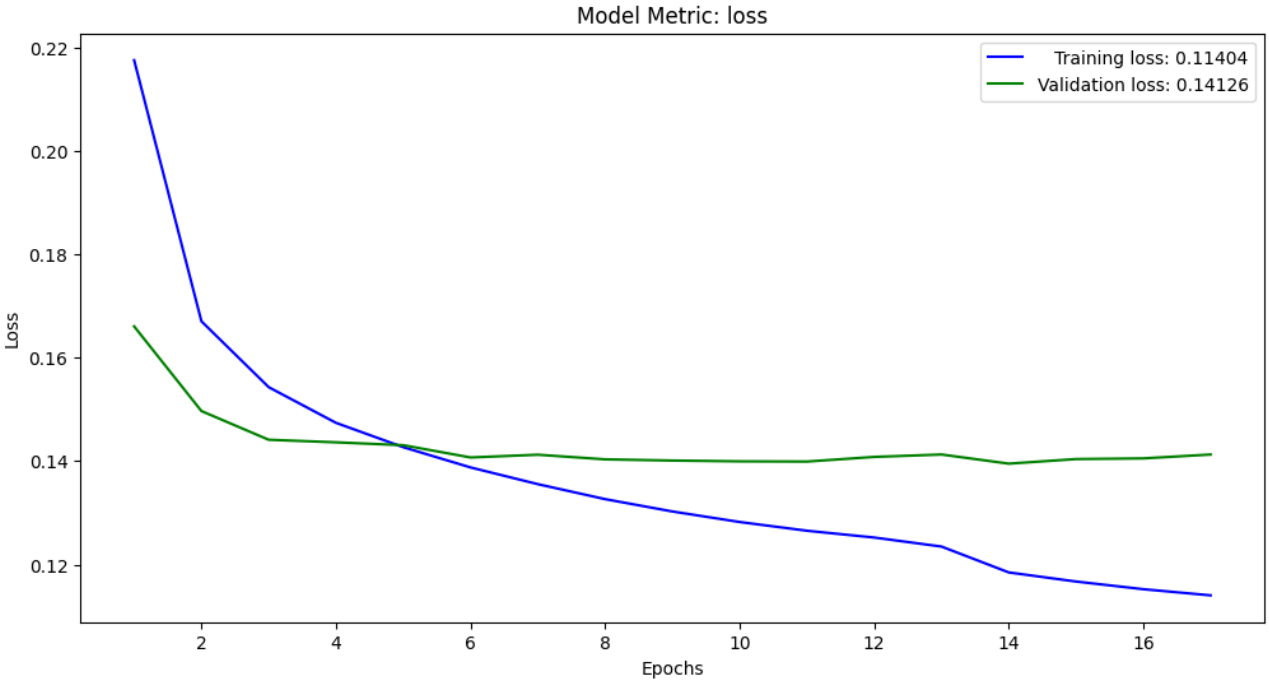


Figure A-2: Train & Validation Precision evolution per epoch of the Sequential model

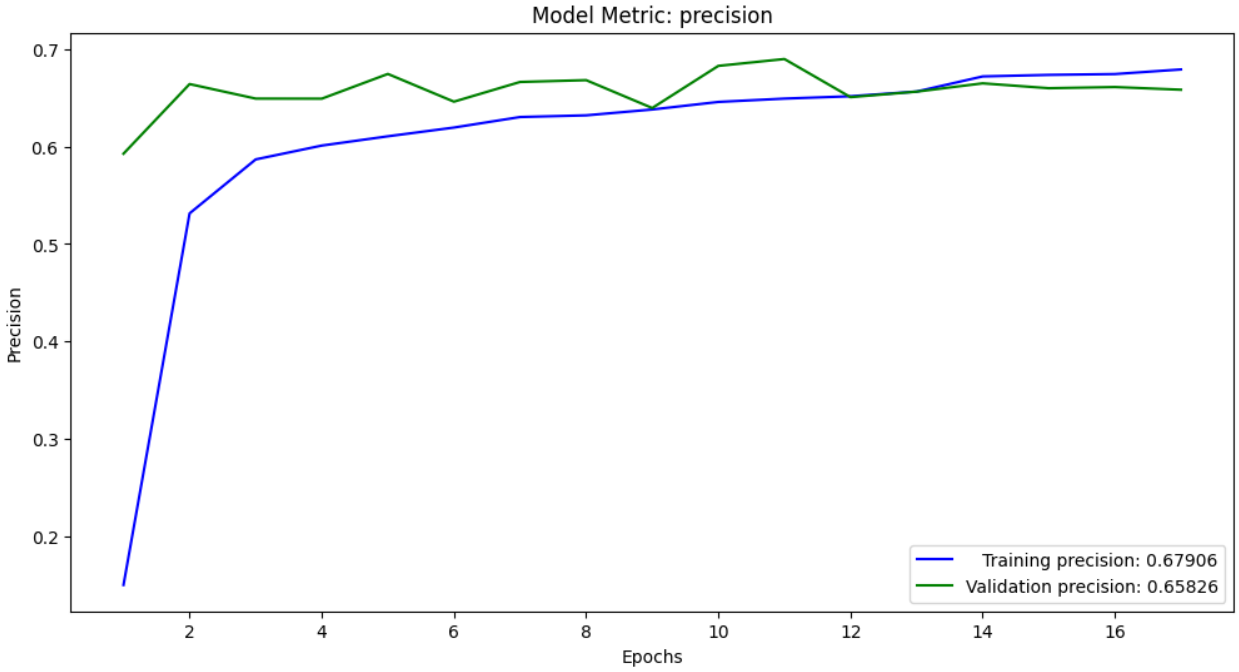


Figure A-3: Train & Validation Recall evolution per epoch of the Sequential model

Figure A-4: Validation Loss evolution per epoch of the DistilBERT model
**(for only 3 epochs)*

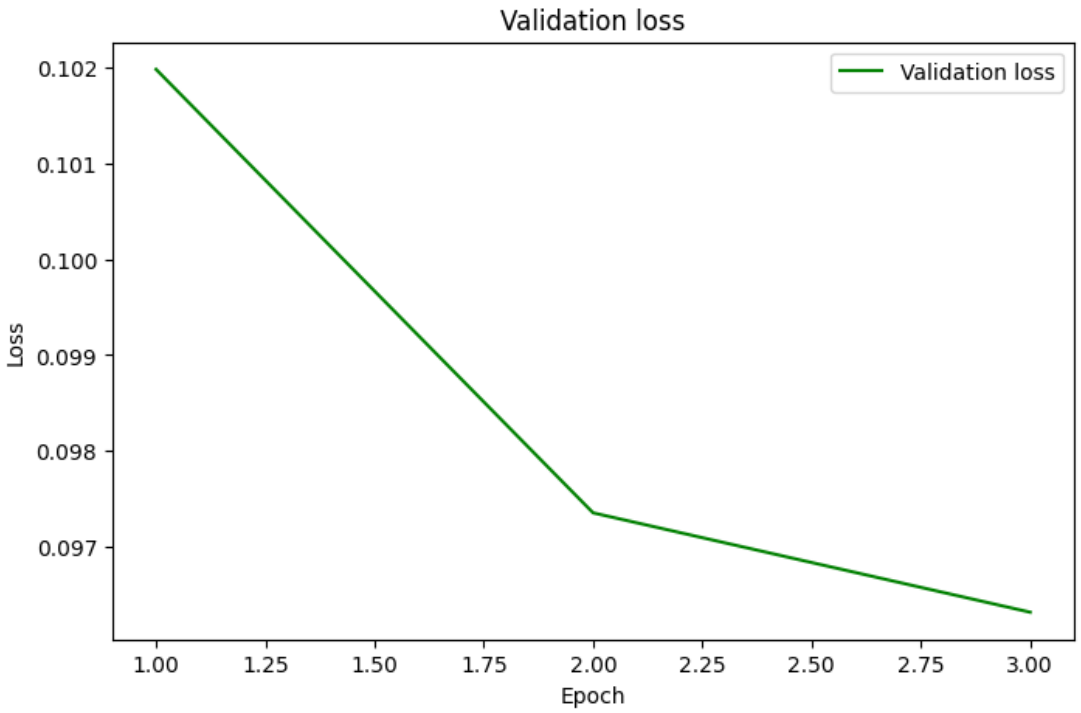


*Figure A-5: Validation Precision evolution per epoch of the DistilBERT model *(for only 3 epochs)*

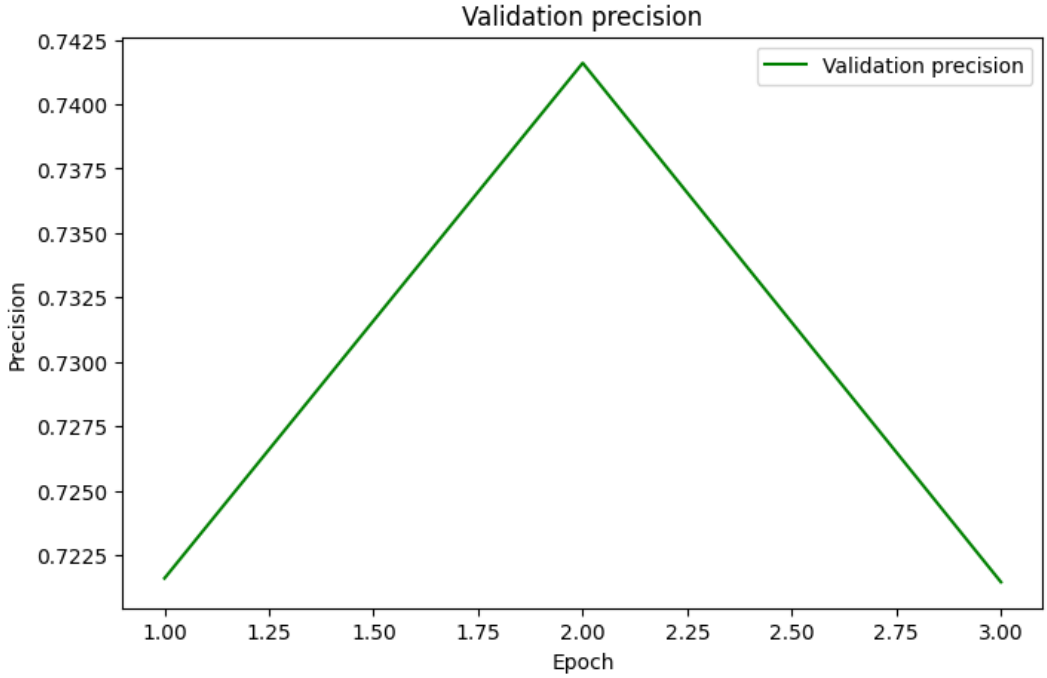


*Figure A-6: Validation Recall evolution per epoch of the DistilBERT model *(for only 3 epochs)*

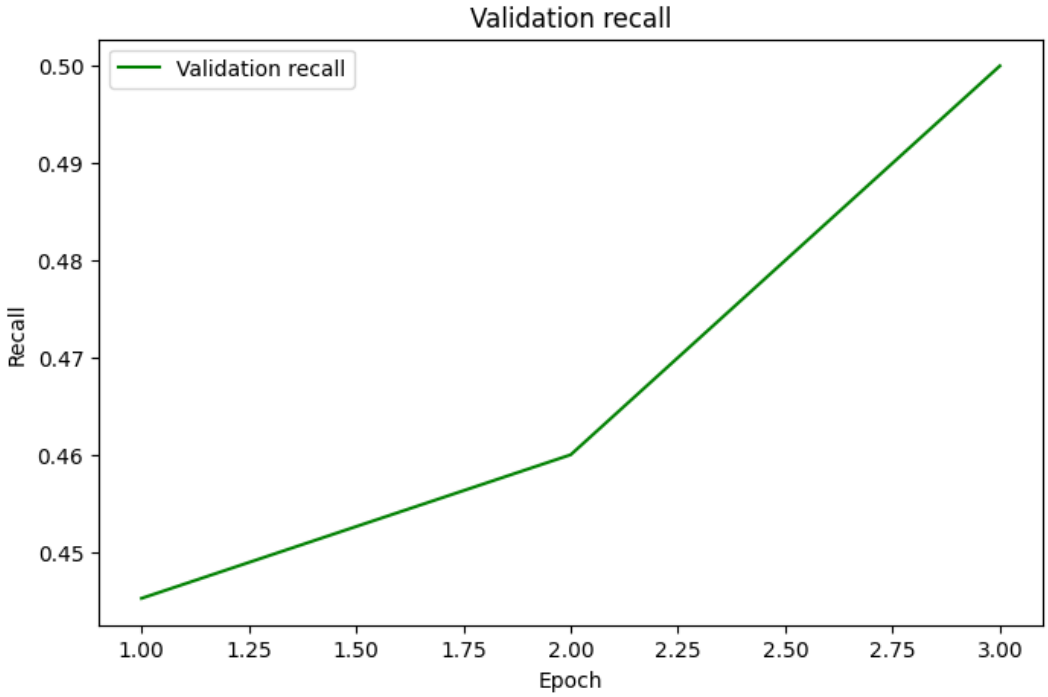


Table A-7: Per Label scores of the Sequential Feed Forwards model

	precision	recall	f1-score	support
Artificial Intelligence	0.00	0.00	0.00	462
Bitcoin	0.55	0.14	0.22	882
Blockchain	0.61	0.45	0.52	1131
Business	0.00	0.00	0.00	581
Covid 19	0.60	0.12	0.20	637
Cryptocurrency	0.60	0.28	0.39	930
Culture	0.25	0.01	0.01	461
Data Science	0.56	0.32	0.41	1124
Design	0.63	0.03	0.05	448
Education	0.58	0.11	0.19	446
Entrepreneurship	0.00	0.00	0.00	419
Health	0.76	0.23	0.35	528
JavaScript	0.63	0.09	0.16	549
Life	0.00	0.00	0.00	867
Life Lessons	0.00	0.00	0.00	689
Love	0.00	0.00	0.00	633
Machine Learning	0.71	0.38	0.50	892
Marketing	0.00	0.00	0.00	442
Mental Health	1.00	0.02	0.05	531
Poetry	0.79	0.56	0.66	962
Politics	0.80	0.46	0.58	745
Programming	0.56	0.13	0.21	969
Python	0.51	0.03	0.06	597
Relationships	0.00	0.00	0.00	502
Self	0.00	0.00	0.00	444
Self Improvement	0.00	0.00	0.00	631
Software Development	0.00	0.00	0.00	589
Startup	0.79	0.04	0.07	745
Technology	0.51	0.05	0.09	967
Writing	0.74	0.16	0.26	788

Table A-8: Per Label scores of the LSTM RNN model

	precision	recall	f1-score	support
Artificial Intelligence	0.61	0.32	0.42	462
Bitcoin	0.71	0.65	0.68	882
Blockchain	0.73	0.74	0.74	1131
Business	0.58	0.06	0.11	581
Covid 19	0.72	0.43	0.54	637
Cryptocurrency	0.63	0.68	0.65	930
Culture	0.58	0.17	0.26	461
Data Science	0.66	0.59	0.62	1124
Design	0.74	0.53	0.62	448
Education	0.71	0.41	0.52	446
Entrepreneurship	0.58	0.16	0.25	419
Health	0.70	0.50	0.59	528
JavaScript	0.80	0.66	0.72	549
Life	0.40	0.00	0.01	867
Life Lessons	0.37	0.03	0.06	689
Love	0.56	0.32	0.41	633
Machine Learning	0.73	0.68	0.70	892
Marketing	0.75	0.47	0.58	442
Mental Health	0.67	0.31	0.43	531
Poetry	0.80	0.75	0.77	962
Politics	0.80	0.67	0.73	745
Programming	0.63	0.46	0.53	969
Python	0.65	0.44	0.52	597
Relationships	0.68	0.32	0.43	502
Self	0.00	0.00	0.00	444
Self Improvement	0.49	0.15	0.23	631
Software Development	0.56	0.25	0.35	589
Startup	0.63	0.33	0.44	745
Technology	0.64	0.22	0.33	967
Writing	0.79	0.50	0.61	788

Table A-9: Per Label scores of the finetuned DistilBERT model

	precision	recall	f1-score	support
Artificial Intelligence	0.65	0.40	0.49	462
Bitcoin	0.74	0.71	0.73	882
Blockchain	0.78	0.76	0.77	1131
Business	0.51	0.17	0.25	581
Covid 19	0.72	0.62	0.66	637
Cryptocurrency	0.63	0.67	0.65	930
Culture	0.59	0.30	0.40	461
Data Science	0.71	0.65	0.68	1124
Design	0.80	0.65	0.71	448
Education	0.75	0.50	0.60	446
Entrepreneurship	0.63	0.23	0.34	419
Health	0.73	0.60	0.66	528
JavaScript	0.83	0.75	0.79	549
Life	0.50	0.06	0.11	867
Life Lessons	0.52	0.05	0.09	689
Love	0.63	0.42	0.50	633
Machine Learning	0.76	0.78	0.77	892
Marketing	0.72	0.57	0.64	442
Mental Health	0.69	0.37	0.48	531
Poetry	0.88	0.77	0.83	962
Politics	0.82	0.70	0.75	745
Programming	0.65	0.55	0.60	969
Python	0.77	0.56	0.65	597
Relationships	0.67	0.41	0.51	502
Self	0.00	0.00	0.00	444
Self Improvement	0.53	0.25	0.34	631
Software Development	0.61	0.35	0.45	589
Startup	0.72	0.42	0.53	745
Technology	0.64	0.28	0.39	967
Writing	0.78	0.61	0.68	788