Analysis and Preprocessing for Youtube Channel Dataset Report

Team 22
Abdelrahman Mohamed Ahmed Abouelkheir 52-5388

March 2025

1 Introduction

In this milestone, we focus on making dataset analysis and pre-processing for unsupervised text classification machine learning task. The goal is to automatically discover and categorize videos within a YouTube channel dataset according to its content without any predefined labels. This choice of task significantly influences what we aim to find during the data analysis phase. and consequently affects the preprocessing steps required to clean and prepare the data for unsupervised learning methods.

The data analysis phase aims to identify the structure of the dataset, understand key trends in the textual data, and highlight potential clusters or topics that could inform our unsupervised classification model and identify inconsistencies through the episodes text and transformation needed to the data. Following this analysis, we will perform preprocessing to ensure the dataset is in a suitable format for machine learning tasks. Preprocessing will include steps such as tokenization, stopword removal, and text normalization, all of which are critical for extracting meaningful features from the data.

Moreover, we also leverage the pretrained model 'faisalq/bert-base-arabic-senpiece' from Hugging Face, use its tokenizer to just visualize and also examine how the data is fed to the BERT based models and see their tokenizer. We also wanted to examine the embeddings produced in the output of the model. We also take those emdeddings and extract classification related data and apply k-means clustering then we visualize the data. This provides an initial understanding of how well the data is grouped into meaningful categories. However, further work is needed in the including fine-tuning the model to improve its performance on this specific dataset. Model fine-tuning will allow us

to better adapt the pretrained model to the dataset and improve the quality of the unsupervised classification results.

2 Dataset Analysis

In this section, we explore the key attributes of the YouTube channel dataset, For each video we have:

2.1 Video Metadata File

The Video Metadata file contains attributes such as the video title, publish date, list of categories, list of keywords, video length, video URL, and author.

Categories: The provided categories list is very limited and generic. Since our goal is to classify the text according to the content of the video, the current categories do not provide enough granularity for accurate classification. We analyze categories and keywords as relate to the task that we want to achieve.

Keywords: We cannot rely on the keywords for the classification task for several reasons:

1. The keywords include items related to the channel itself.
2. They also contain items about the organization that owns the channel.
3. These items are common across all videos since they belong to the same channel, making the keywords too generic.
4. If we attempt to clean the data by removing irrelevant keywords, we would be left with only a small number of usable keywords.

The keywords are either too specific (leaving insufficient data) or too generic (grouping all videos into the same category). As a result, we believe the metadata's keyword information will not be useful for text categorization that we want to do.

2.2 Raw Data

Raw Data the text data for the content of video. We will rely on this file to achieve the task, thereby we need to make some analysis of this text to decide what we will do in the pre-processing phase.

1. **Inconsistency in Writing:** The text contains several inconsistencies. For example, some text use ة when needed , while other text replace it with ه for the same words. Similarly, some text include أ , while others omit the ء, leading to inconsistency. Furthermore, there is inconsistency

between the use of Alef Maksura (ى) and Yeh (ي) in certain words. This lack of uniformity needs to be addressed.

2. **Intro and Outro Phrases:**

   The author includes introductory and concluding phrases at the beginning and end of the text. These phrases are common across different videos.

3. **Incorrectly/incomplete Written Words:** Some words in the text are written incorrectly or incomplete.

4. **Egyptian Arabic with English Words:** The majority of the text is written in Egyptian Arabic, but there are small portions of English words, often used for company names, people's names, or scientific terms related to the topic. These English words are important as they provide context and information about the topic and should be preserved.

5. **Stop Words:** The text contains many stop words, such as لكن, هو, and هي, which do not contribute to the meaning of the content and can introduce noise. These stop words should be removed.

6. **Isolated Letters:** Some parts of the text contain isolated letters with no meaning, which should also be removed as they introduce unnecessary noise.

7. **Unwanted Unicode Characters:** There are certain Unicode characters present in the text that have no meaning and need to be removed.

8. **Bracketed Words:** The text contains words enclosed in brackets, such as [موسيقى] , which are usually references to sounds or actions rather than meaningful text. Since there is no mathematical content that justifies keeping these, all words between brackets should be removed to reduce noise.

9. **Newline Characters and Space Alignment:** The text may contain newline characters ('\n') that break the flow of words but i does not act as a separator between sentences. These characters should be removed, and spaces between words should be aligned properly after removing them.

Note: There are other files in the dataset but they are just for guiding.

3 Pre-processing and Data Cleaning

To prepare the data for modeling, we perform the following preprocessing steps:

3.1 Data Cleaning

Data cleaning is a critical part of the pre-processing phase to ensure that the raw text is free of noise and inconsistencies. Several steps are performed to clean the text:

1. **Removing Bracketed Words:** Some words in the text are enclosed in square brackets, such as [موسیقی], which usually represent non-verbal elements like sounds or actions. These bracketed words, along with the brackets, are removed from the text to prevent them from introducing noise.

2. **Removing Extra Spaces:** The text may contain multiple consecutive spaces or unwanted spaces between sentences. These extra spaces are removed to ensure consistent spacing between words.

3. **Removing Sentence Spaces:** Sometimes, unnecessary spaces appear between words at the start of a sentence or after punctuation marks. These spaces are cleaned up to align the text properly.

4. **Removing Isolated Letters:** The text might contain letters standing alone without context, which have no meaning in the content. These isolated letters are identified and removed to avoid adding unnecessary noise to the text.

5. **Removing Unwanted Unicode Characters:** Some Unicode characters present in the text may not contribute meaning to the content and can introduce inconsistencies in the model. These unwanted characters are removed to make the text cleaner and more uniform.

3.2 Normalizaion

In this function, several operations are performed using packages from pyarabic and camel-tools, which are widely used for processing Arabic text. Below is an explanation of each step:

- **Tashkeel Stripping:** We use the function `strip_tashkeel` from the `pyarabic.araby` package to remove diacritics (tashkeel) from the text. Tashkeel includes marks such as ٱ أ إ that are used to indicate vowel sounds. Removing tashkeel is important because the same word may be written with or without diacritics, and we want to avoid treating them as different words.
- **Small Harakat Removal:** The function `strip_small`, also from `pyarabic.araby`, removes small Arabic characters that are used for elongation or special sounds, which are not necessary for our analysis and may introduce noise.
- **Alef Normalization:** Using the `normalize_alef_ar` function from the `camel_tools.utils.normalize` package, we standardize different forms of Alef characters (such as ا, أ, إ, and آ) into a single form. This step ensures that variations in Alef spelling do not affect the text processing.
- **Alef Maksura Normalization:** The function `normalize_alef_maksura_ar` from the `camel_tools.utils.normalize` package normalizes occurrences of the Alef Maksura character (ى) to Yeh (ي). This is necessary because both characters may represent the same sound in different contexts.
- **Teh Marbuta Normalization:** Using the `normalize_teh_marbuta_ar` function, we normalize Teh Marbuta (ة) to Heh (ه). Teh Marbuta and Heh are sometimes used interchangeably at the end of words, and normalizing them helps maintain consistency in the text.
- **Author-Related Text Removal:** The text often contains repetitive patterns such as the author's name or introductory and concluding sentences that appear in every video. These patterns are removed using custom functions like `removeAuthorStartSentence`, `removeAuthorEndSentence`, and `removeAuthorName` to prevent them from influencing the classification of video content.

3.3 Tokenization

Tokenization is a crucial step in natural language processing (NLP), where the input text is divided into smaller units known as tokens, which are typically words or subwords. This process allows us to work with each token individually in the subsequent stages of text analysis, such as feature extraction or text classification.

For tokenizing the Arabic text, we use the `pyarabic` library, specifically the `araby.tokenize` function. This function is designed for handling Arabic text, splitting it into individual tokens while taking into account the unique characteristics of the Arabic script.

Here is what this process entails:

- **Arabic-Specific Tokenization:** Unlike other tokenizers, `araby.tokenize` is tailored for Arabic text, which has specific challenges such as handling diacritics, complex word forms, and different word boundaries. This ensures more accurate tokenization when working with Arabic language text.
- **Efficient Tokenization Loop:** We loop through the preprocessed (normalized) text data using the `tqdm` library to keep track of progress and tokenize each text file individually. This step produces a list of tokenized sentences stored in `tokenizedDataPhaseOne`.
- **Tracking Progress:** We employ `tqdm`, a Python library for creating progress bars, to provide real-time feedback during the tokenization of multiple text files, ensuring the process is transparent and trackable.

3.4 Stop Words Removal

Stop words (such as هو and أنا) are words that carry little semantic value and may introduce noise during classification. We remove stop words after normalizing the text to ensure consistent representation. The stop word list was sourced from a GitHub repository containing standard Arabic stop words. Additionally, we extended the list to include Egyptian Arabic stop words commonly found in our dataset. Before applying stop word removal, we also normalize the stop words themselves to ensure they match the format of the text.

3.5 Stemming

Stemming is the process of reducing words to their root form by removing affixes like suffixes and prefixes. The goal of stemming in text preprocessing is to treat

different variations of a word as the same word. For example, the Arabic words كتب, كاتب, and كتبت can all be reduced to the same root كتب. This helps reduce the vocabulary size and improves the accuracy of models by reducing the noise caused by word variations.

For stemming Arabic text, we tested two stemmers: **Farasa Stemmer** and **ISRIStemmer**. After comparison, we found that the **ISRIStemmer** provided more accurate results than the Farasa Stemmer for our dataset and the chosen pretrained model. The **ISRIStemmer** is part of the NLTK library and is tailored specifically for Arabic, offering a basic and efficient method for stemming Arabic text.

By reducing words to their stems, we aim to enhance the ability of our classification model to generalize and better identify the core meaning of each sentence.

3.6 BERT Model Tokenization and Clustering

In this section, we utilized a pre-trained BERT model to create embeddings for our Arabic text data. The model used was **faisalq/bert-base-arabic-senpiece**, which is a version of BERT fine-tuned for the Arabic language using the SentencePiece tokenizer. This model was specifically chosen due to its effectiveness in representing Arabic language text for downstream tasks, providing robust embeddings that are suitable for tasks such as classification, clustering, and similarity measurement.

BERT (Bidirectional Encoder Representations from Transformers) is a transformer-based model known for its capacity to capture deep contextual relationships between words in a sentence. It processes text bidirectionally, meaning it reads the input from both directions (left to right and right to left) to understand the context more effectively. This helps in capturing the full semantic meaning of words in relation to their surrounding text.

The first step in the process was tokenization. We employed the **AutoTokenizer** from the Hugging Face Transformers library, which efficiently tokenized the text while ensuring padding for uniform length, truncation of longer sequences, and conversion into the required format for the BERT model. Tokenization is essential because it breaks down the text into smaller units (tokens) that the model can understand and process.

Once the tokenization was complete, we passed the tokenized text into the pre-trained **BERT model** to obtain the embeddings. The embeddings generated by BERT are dense vector representations of the text, capturing both syntactic and semantic properties of the input. Specifically, we used the embeddings from the [CLS] token of the BERT model, which is commonly used for classification tasks. These embeddings serve as a condensed representation of the entire sentence.

Next, we applied **K-Means clustering** to the generated embeddings. Before clustering, we normalized the embeddings using **StandardScaler** from the scikit-learn library, ensuring that each feature contributed equally to the cluster-

ing process. Clustering is the process of grouping similar data points together. The K-Means algorithm partitions the data into k clusters, with each data point belonging to the cluster with the nearest mean. This allows us to group similar sentences together based on the semantic properties captured by the BERT embeddings.

The number of clusters was chosen based on evaluation, and the final clusters represent different groups of semantically similar sentences. This approach helps us to understand the latent structure of our dataset, which can be further utilized in various downstream tasks such as content categorization, search optimization, or classification.

Tools and Libraries Used:

- **Hugging Face Transformers:** This library provides an extensive collection of pre-trained models, including BERT, and tokenizers for various languages. It is widely used for Natural Language Processing (NLP) tasks.
- **PyTorch:** We used PyTorch to load and perform inference with the BERT model. PyTorch is an open-source machine learning framework that enables flexible experimentation with deep learning models.
- **scikit-learn:** A comprehensive machine learning library that includes tools for data preprocessing, clustering (K-Means in our case), and other machine learning algorithms.
- **NumPy:** Used to manipulate arrays and convert PyTorch tensors into arrays for further processing.

In conclusion, by combining a pre-trained BERT model with K-Means clustering, we successfully captured the semantic relationships between sentences in our dataset, allowing for meaningful clustering that can be used for categorization and other related tasks. This approach leverages state-of-the-art tools and models to process and analyze Arabic text efficiently.

4 Conclusion

In this milestone, we performed detailed data analysis and preprocessing of a YouTube channel dataset, preparing the data for NLP modeling. Using a pre-trained BERT model, we extracted feature embeddings and explored their use in clustering and classification tasks. The next step would involve further model tuning, experimentation with additional models, and extending the analysis to broader applications like sentiment analysis and topic modeling.

5 References

1. Mdpi, "Arabic Sentiment Analysis Using Pre-trained Language Models: A Comparative Study," Available: `https://www.mdpi.com/2076-3417/14/13/5696`.

2. Hugging Face Transformers, "faisalq/bert-base-arabic-senpiece," Available: https://huggingface.co/faisalq/bert-base-arabic-senpiece.

3. Hugging Face, "Transformers Library," Available: https://huggingface.co/transformers/.

4. PyTorch, "PyTorch Framework," Available: https://pytorch.org/.

5. scikit-learn, "Machine Learning in Python," Available: https://scikit-learn.org/.

6. NumPy, "NumPy Documentation," Available: https://numpy.org/.

7. Camel Tools, "camel-tools Python library," Available: https://github.com/CAMeL-Lab/camel_tools.

8. nltk, "Natural Language Toolkit," Available: https://www.nltk.org/.

9. pyarabic, "PyArabic Python library," Available: https://pypi.org/project/PyArabic/.