M915 NLG & NLU (Fall Semester 2023)

Assignment 1: Summarization System Comparison

Report

Anastasia Karakitsou (It12200009)

Kleopatra Karapanagiotou (It12200010)

Both authors contributed equally to this project.

Intro/Objectives

The aim of Assignment_1 is to build 3 summarizations systems, by following 3 different approaches:
1) use as a baseline a simple Lead-2 summarizer,
2) use machine learning techniques in order to extract n summary worthy sentences from each article, and
3) experiment with existing large language models and decide on an approach on how to use a large language model for summary generation.

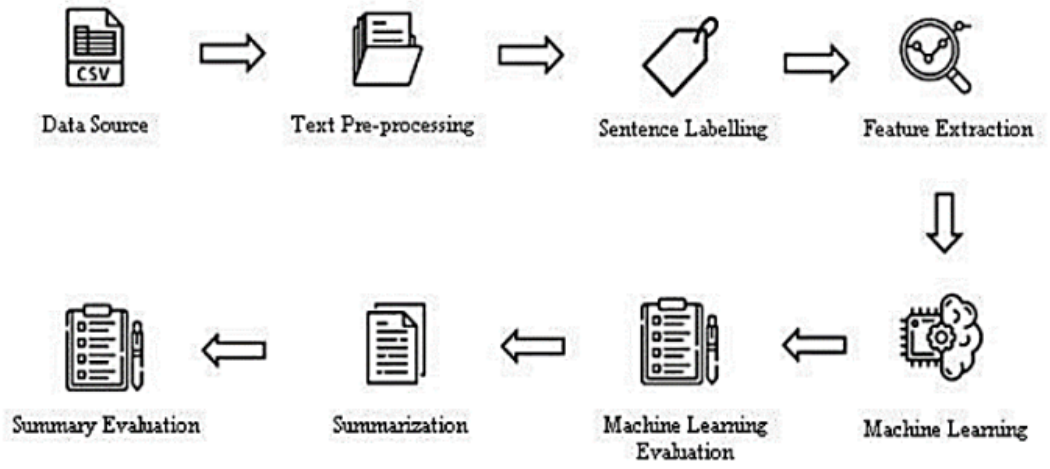
System 1 : Lead-2 Summarizer

This summarizer is designed to extract and return the initial two sentences from each input text. The process involved tokenizing each original article into sentences, from which the first two were then extracted. These extracted sentences were later compiled and stored in a file named 'lead_2_summs.txt'. Considering the assumption that news articles have been shown to conform to writing conventions, in which important information is primarily presented in the first one-third of an article, we are confident that our baseline system will give a satisfactory baseline score.[1]

System 2 : Machine-Learning based Extractive Summarizer

Problem Understanding

In Machine Learning approaches the summarization task is modeled as a classification problem, where **sentences** are classified as **summary sentences [1]** and **non-summary sentences [0]** based on the features they possess. The following image[2] captures all the stages implemented in our text summarization modeling approach:



[1] Information about article conventions found here: https://huggingface.co/datasets/cnn_dailymail
[2] Image taken from Jian et al. (2022), p.181

Data Acquisition

The dataset used in our experiments, the CNN/DailyMail v3 consists of news articles and highlight sentences. It was downloaded from Hugging Face and transformed to .csv form for partitioning into smaller portions and for the application of next preprocessing steps. All Machine Learning experiments were conducted with the 10% of the train and validation data, whereas for summary generation the test set was used in its entirety.

Data Exploration

This basic step gave us important insights for the CNN V3 dataset, in particular the train and validation datasets (NB: we avoided exploring the test dataset, so that it remains 'unseen', and so that we remain unbiased. Also, considering that the test set was taken from the same context, we expected it to exhibit similar data patterns). You may view the patterns in the two tables below:

	Total entries	*1% (only for experimenting)*	*10% (for final training, only for train and validation)*
Train	287,113	287	28,711
Validation	13,368	134	1,337
Test	11,490	115	N/A

	Average number of sentences	*Average number of words*	*Total number of words*
Articles	approx. 38.8	approx. 787	236,537,429
Highlights	approx. 3.7	approx. 55	16,513,322

Data Preprocessing

We divided our cleaning pipeline into three steps:

Steps	**Description**
Removal of Internet clutter	URLs, HTML, https:/ and href tags, @usernames
Removal of boiler-plate and	● '(CNN)--' and the preceding words at the beginning (only for articles)

specific patterns	● ‘NEW:’ at the beginning (only for highlights) ● ‘E-mail to a friend’ and anything that follows ● ‘ (_) ’: Text inside parenthesis, which included descriptions of accompanying videos, photos, and ads
Text standardization and cleaning irrelevant linguistic features (spaCy)	Removal of non-ASCII characters, perform lowercasing, expand contractions, tokenization, removal of stopwords & punctuation, lemmatization

Even though we ended up performing rigorous cleaning, we decided not to use the dataset resulting from the final step, as that would cause problems in calculating accurate word embeddings. The reason is that it is generally more effective to use the original tokens rather than the lemmatized ones, since word embeddings are typically trained on large corpora of text in their original form, and using lemmatized tokens might lead to a loss of semantic information. Therefore, we decided to use the cleaner version of our articles and highlights, resulting from the second cleaning step, since this version is more concise, more relevant to the actual article content, and also less noisy.

Sentence Labeling

We performed a sentence-based tokenization for both article and highlights, then we exploded the dataframes, so that each article sentence is located in a separate row. We calculated the mean word embeddings of the separated article sentences as well as the highlights using the pre-trained model ‘word2vec-google-news-300’, which we selected based on its relevance to our genre. After we had extracted the mean sentence embeddings, we proceeded to compute the cosine similarity scores between each article sentence against all of the sentences present in the corresponding highlights, with the view to discovering which article sentences were used for the creation of the highlights. Then we selected the maximum cosine similarity score for each article sentence in order to create its label: for every article, we selected *four* sentences with the highest similarity scores and gave them a label of 1, while the rest were given a label of 0. We chose to label *four* sentences with the view to teaching the classifier to create predictions of *four* highlights, based on the average sentence count in our dataset’s highlights, which was approx. 3.7. We therefore structured our dataset in an imbalanced way and experimented with hyperparameter tuning on the Logistic Regression algorithm.

Feature Extraction

First, we used the average word2vec embeddings of each article sentence as a type of content feature, as well as the maximum cosine similarity score obtained through the process of sentence labeling, since this score denotes the connection of each sentence in the article with one of the sentences in the highlights. We further extracted two separate surface features, such as sentence position (based on the total amount of sentences, where 1 was the maximum score for the first sentence, 0 for the very last one, and all the remaining sentences were given equally spaced scores), and sentence length (normalized against the maximum sentence length for each article).

Model Development/ Machine Learning Experiments

Our experiments were carried out with a 12th Gen Intel(R) Core(TM) i7-12700H 2.30 GHz CPU, on an ASUS TUF GAMING F15 laptop with 16.0 GB RAM using a Windows 11 Home OS.

After scaling our input features and labels, we used a Logistic Regression model, since it is an efficient and popular binary classification model, and because we assume our data is linearly separable into the two classes (summary-worthy class=1 Vs. summary-unworthy class=0). First, we performed a parameter grid search using our experimental train and validation datasets in order to obtain the optimal hyperparameters based on precision score. Our first grid search did not include any specific class weighting. Below you may find the results for the non-weighted model:

```
Best hyperparameters during the grid search: {'C': 0.1, 'max_iter': 500, 'penalty': 'l1', 'solver': 'saga'}
```

In [32]:
```
y_val_pred = best_clf.predict(X_val_scaled)

# Generate the classification report
report = classification_report(y_val_np, y_val_pred)
print("Classification Report (Validation Set):")
print(report)
```

```
Classification Report (Validation Set):
              precision    recall  f1-score   support

           0       0.96      0.97      0.96      4108
           1       0.73      0.69      0.71       536

    accuracy                           0.94      4644
   macro avg       0.85      0.83      0.84      4644
weighted avg       0.93      0.94      0.93      4644
```

We can observe that the precision for the summary-worthy class is 73%, its recall is 69%, and its f1-score reaches a 71%, which means that all scores pertaining to class 1 are pretty decent.

Our second grid search included specific values for class weighting. Below you may view the results for the weighted model:

```
Best hyperparameters during the grid search: {'C': 0.1, 'class_weight': {0: 0.565, 1: 4.332}, 'max_iter': 500, 'penalty': 'l1',
'solver': 'saga'}
```

In [26]:
```
y_val_pred = best_clf.predict(X_val_scaled)

# Generate the classification report
report = classification_report(y_val_np, y_val_pred)
print("Classification Report (Validation Set):")
print(report)
```

```
Classification Report (Validation Set):
              precision    recall  f1-score   support

           0       0.99      0.83      0.90      4108
           1       0.41      0.91      0.57       536

    accuracy                           0.84      4644
   macro avg       0.70      0.87      0.73      4644
weighted avg       0.92      0.84      0.86      4644
```

Surprisingly, we observe that the score for precision is significantly lower, at 41%. Since precision is very important to us, i.e. the minimization of FPs, we naturally chose the hyperparameters of the first grid search for our actual training.

Similar disappointing results were observed also after downsampling the majority class [0] with a 50-50 ratio, to tackle the class imbalance. Below the results on the validation set.

```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.81      0.89      4108
           1       0.39      0.93      0.55       536

    accuracy                           0.83      4644
   macro avg       0.69      0.87      0.72      4644
weighted avg       0.92      0.83      0.85      4644


ROC AUC Score: 0.8694538505137409

Confusion Matrix:
 [[3342  766]
 [  40  496]]
```

Machine Learning Evaluation

The actual training, where the 10% of the train and the 10% of the validation datasets were used, resulted in the following scores when evaluated in the validation set, and look very promising:

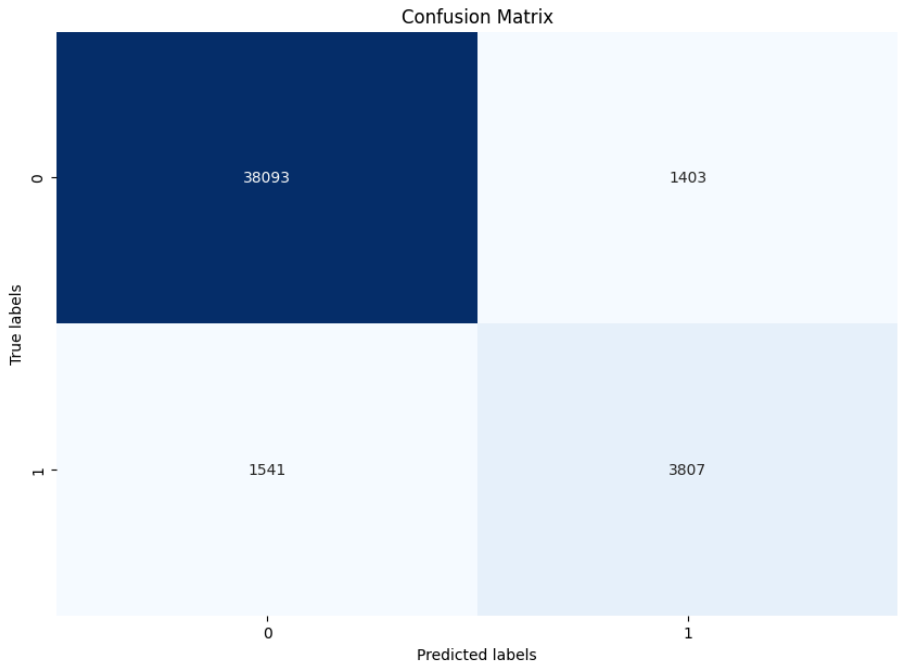
```
Classification Report (Validation Set):
              precision    recall  f1-score   support

           0       0.96      0.96      0.96     39496
           1       0.73      0.71      0.72      5348

    accuracy                           0.93     44844
   macro avg       0.85      0.84      0.84     44844
weighted avg       0.93      0.93      0.93     44844
```

As we can observe, the summary-worthy class exhibits a precision score of 73%, a recall score of 71%, and a f1-score of 72%, which are all very satisfactory for our case, considering we have an imbalanced class.

Below you may view the confusion matrix. We can observe that the TPs for the summary-worthy class were 3,807 with an almost equal number of FPs (1,403) and FNs (1,541).



A further metric we used is the ROC-AUC score, which was as high as 0.84. This value indicates that our final model has a good ability to distinguish between the two classes. In particular, it has a 83% chance to correctly distinguish between a class 1, and a class 0 sample.

Summary Generation

We concatenated the classifier's predictions onto the modified dataframe with the article sentences and their corresponding highlights column-wise. Based on the unique article ID we extracted the first three sentences with the class 1 prediction, and wrote them onto a txt file. We chose to write three sentences for each predicted summary in order to make our model more strict. After extracting the highlights, we then passed the two files from the ROUGE-2 scorer. It is worth noting that the comparison between predicted and golden summaries was conducted on the tokenized, boiler-free text, in which punctuation characters are also space separated.

System 3: LLM-based Summarizer

Bert-extractive-summarizer

We first used a ready tool that utilizes a neural network approach called BERT to run extractive summarizations[3]. It works by first embedding the sentences with BERT, then running a clustering algorithm, finding the sentences that are closest to the cluster's centroids. In our first LLM experiment we simply use the whole test set of the CNN/DailyMail dataset and extract the

[3] https://pypi.org/project/bert-extractive-summarizer/

3 most informative sentences for each article based on the summarizer model predictions. No extra parameters are specified like applying coreference resolution or minimum summary length. Inference has been conducted on the whole test set.The results in terms of Rouge-2 metrics are quite low, which is though an expected behavior, since no training or finetuning has been applied.

Fine-tuning a Simple T5-base for Abstractive Summarization

Our second approach in terms of harnessing the power of LLMs to build a summarizer was to finetune a **T**ext-**t**o-**T**ext **T**ransfer **T**ransformer (T5) model. T5 considers all aspects of the transfer learning pipeline in NLP, such as different (unlabeled) datasets, pre-training objectives, benchmarks and fine-tuning methods. The goal of T5 is to *i)* analyze transfer learning setups and *ii)* determine the most effective approaches.

Considering the sequence-to-sequence architecture of T5 we frame our task for this experiment as abstractive summarization task, in which we expect the fine-tuned model to generate a paraphrased concise version of the original article, rather that expecting the model to cut and paste sentences of the original article to form an extractive summary. We are aware of the limitations of rouge2 score in terms of bigram overlap, but considering that the highlight content (the golden summaries) has been written from the human article authors, we are confident that an abstractive text generation will give us another view of the dataset and the summarization approaches on it. In terms of model size we use the T5_base model , which is trained on the Common Crawl's web crawl corpus (C4)[4], a 750Gb corpus of “relatively clean” English text. To be able to run experiments with our available computational resources we limit the train data for fine tuning to 5,000, out of which 1,500 were separated for validation. In both LLM experiments we use the boiler-free version of the articles and highlights text. Due to lack of computational units, inference has been applied on half of the test set (~5,500 articles).The hyperparameters of the fine-tuning can be summarized in the following table:

```
MAX_LEN= 512 (max length of input sequences-articles, in tokens)
SUMMARY_LEN=150 (max length of generated sequences-summaries, in tokens)
MAX_EPOCHS=5
batch_size=8
use_gpu=True
```

Fine-tuning took around 1 hour, whereas inference on the half test set and calculation of Rouge-2 metrics took approximately 2.5 hours.

[4] https://huggingface.co/datasets/c4

Evaluation

ROUGE

(Recall-Oriented Understudy for Gisting Evaluation) is the common method for assessing the quality of machine generated summaries over human generated ones, since it correlates positively with human evaluation, it is inexpensive to compute, and language-independent. The main drawback which should be taken into account for the abstractive summaries is that it does not capture different words that have the same meaning, as it measures syntactical matches rather than semantics.

ROUGE-2

Given a reference R and a candidate C summary, ROUGE-2 precision is the ratio of the number of bi-grams in C that appear also in R, over the number of bi-grams in C. ROUGE-2 recall is the ratio of the number of 2-grams in R that appear also in C, over the number of bi-grams in R.

Overall System Comparison

Systems	**Rouge-2 scores**	
Lead_2 (Baseline)	Mean Precision: 0.166 Mean Recall: 0.159 Mean F-measure: 0.156	Std Precision: 0.125 Std Recall: 0.121 Std F-measure: 0.112
ML-extractive (feature-based)	Mean Precision: **0.205** Mean Recall: **0.299** Mean F-Measure: **0.235**	Std Precision: 0.132 Std Recall: 0.165 Std F-Measure: 0.134
Bert-extractive-summarizer (pre-trained model usage on test set)	Mean Precision: 0.088 Mean Recall: 0.131 Mean F-Measure: 0.102	Std Precision: 0.083 Std Recall: 0.113 Std F-Measure: 0.089
T5_base (fine-tuned)	Mean Precision: 0.171 Mean Recall: 0.182 Mean F-Measure: 0.170	Std Precision: 0.124 Std Recall: 0.131 Std F-Measure: 0.117

Lead_2: Given the mean and standard deviation values in all three Rouge-2 metrics, we see that the generated summaries have an average 16% of bigrams also found in the gold summary, meaning that the Lead_2 baseline system has a consistent low performance. This was expected behavior, since the Lead_2 system extracts the first 2 sentences of the articles, whereas the 'highlights', which are considered the gold summaries in our data, capture several points throughout the article.

ML_extractive: We see that it outperformed all rest experiments. By maintaining a ratio of 90-10 in the class distribution our logistic regression algorithm managed to extract the most informative sentences, based on the cosine similarity scores. Again it is important to mention that Rouge-2 scores were calculated on the tokenized boiler-free text of both predicted summaries and highlights, where punctuation characters were also tokenized and space separated.

BERT-extractive: The use of a ready system was an initial naive approach to evaluate sentence transformer embeddings in the extractive summarization process. It is obvious that this model doesn't even reach the baseline, leading us to use other techniques like fine-tuning LLMs to be able to reach a decent Rouge-2 score.

T5: The slightly higher than the baseline performance of the fine-tuned T5 can be explained due to the few amount of train data used, the few epochs, and batch size used, as well as due to the diverse nature of the CommonCrawl data that T5 was pre-trained on. Also it is worth noting that inference for the T5 model experiment was conducted on half of the test data, due to lack of computational resources. We expect that with more train data and more epochs and batch size the T5 abstractive summarizer will outperform all tested experiments above.

Considerations for Future Work

A. Machine Learning

Model exploration: To address the issue of class imbalance several other machine learning algorithms can be tried, like SVM, Random Forest, Naive Bayes, Decision trees. Furthermore, ensemble methods like Adaptive Boosting can be tried in order to sequentially harness the power of simpler classifiers and achieve a higher performance. Ensemble methods can be very effective due to their ability to capture complex interactions between features, and their robustness against overfitting.

Feature selection: Due to computational power and time constraints, our project could not incorporate certain features in the final training pipeline. Notably, features like normalized counts of named entities and verbs showed promise during our initial 1% experimentation phase. Future work should focus on integrating these features to observe their impact on the classifier's performance.

Mean Word2Vec embeddings vs Sentence Embeddings: It is common knowledge that static embeddings (like Word2vec) do not consider word order, meaning that two sentences like the following:

• our president is a good leader he will not fail
• our president is not a good leader he will fail

will have the same mean word embedding, even though it is obvious that these two texts have the opposite meaning. This is why implementing advanced sentence representation models, such as sentence-BERT, could significantly improve the reliability of cosine similarity calculations, offering a more nuanced understanding of the sentences.

B. LLMs

Optimize fine-tuning parameters: We discussed opportunities to enhance the current fine-tuning approach so as to improve the model performance by experimenting with larger batch sizes, increased training data size, and extended training epochs.

Exploring Diverse Model Architectures: For text generation tasks, particularly in abstractive summarization, experimenting with various model sizes and architectures could offer interesting results. Larger models like T5-Large and BART-Large are great choices for abstractive approaches. Also, more compact models like DistilBERT could be explored for both extractive and abstractive summarization techniques.

Github link: https://github.com/akitsou/summarization-models.git

References

Text Summarization with Large Language Models. *Artificial Intelligence in Plain English.* Retrieved from https://ai.plainenglish.io/text-summarization-with-large-language-models-c9ae4be96863 . Last accessed: 12 Jan 2024.

Main Classes: Trainer. (2023). Hugging Face's Transformers Documentation. Retrieved from https://huggingface.co/docs/transformers/v4.34.1/en/main_classes/trainer . Last accessed: 12 Jan 2024.

Jian, H., Johnson, O., & Wah, K. (2022). Text Summarization for News Articles by Machine Learning Techniques. *Applied Mathematics and Computational Intelligence*, 11(1). Retrieved from http://dspace.unimap.edu.my/bitstream/handle/123456789/77725/Text%20Summarization%20for%20News%20Articles%20by%20Machine%20Learning%20Techniques.pdf?sequence=1&isAllowed=y . Last accessed: 12 Jan 2024.

Wong, K. F., Wu, M., & Li, W. (2008). Extractive Summarization Using Supervised and Semi-Supervised Learning. *Proceedings of the 22nd International Conference on Computational Linguistics* (Coling 2008), pp. 985-992. Retrieved from http://anthology.aclweb.org/C/C08/C08-1124.pdf . Last accessed: 12 Jan 2024.

Fine-Tune SimpleT5 Text Summarization. (n.d.). *Kaggle.com*. Retrieved from https://www.kaggle.com/code/psugunnasil/961791-fine-tune-simplet5-text-summarization . Last accessed: January 12, 2024.