A Survey on Usage of Vectorizers for Textual Data in Exploratory Data Analysis (EDA)-Generative AI

DEVARAM SHARATHCHNDRA
Software Engineer, Machine Learning Enthusiast

Abstract*— In the early age of Artificial Intelligence (AI) and Machine Learning (ML) domain, mostly training of ML models are depended on numerical data to classify, predict or generate. In today's world we achieved the state "Machine models" can interact with human in a pure form of humanized text. Natural Language Processing (NLP) is the growing domain where it interacts with human in a way of speech recognition, text classification and text generation. The present era is experiencing prompt-based AI, where we can generate new images with a simple text prompt input or can generate a professional video or chat bot types models for virtual assistance. Simultaneously we are interacting with speech with a machine. The core technology behind this textual input is vectorizing the text data. When we interact with ML model with a speech input, in the background-the speech is converted into a textual format and then vectorized for prediction or generation to produce output. Based on the produced output the output layer can interact with human according to the choice provided by the end user weather it is belonging to NLP or Text generation transformer type model. The best example for humanized text generation model we are experiencing in today's technology era are Google's Gemini and Open AI's Generative Pre-Trained Transformer (GPT) model. Vectorizers are the main technology behind these text transformation and analyzation models. The main amin these vectorizers re to improve machine learning model accuracy and reducing computational complexity of a ML model. NLP use multilayered neural networks for a Deep Learning (DL) model. Before feeding the first input layer with this textual data, we are using this vectorizers concept while training the deep learning model. Vectorization concept is involved in feature extraction and these will include different type of vectorizers. In this survey paper we discussed most of the vectorizers in section wise. In the I. Introduction section, I am going to introduce the concepts of vectors and what are different types of vectors available to use for machine learning model. From the section II. Core Technology, I'll explain how we use vectorizers for a Machine Learning, Deep Learning and Transformer models to train. From the final section III. Results, difference between all type of vectorizers are concluded.

***Index Terms*—Vectorizers, Machine Learning (ML), Deep Learning, NLP, Transformers, Artificial Intelligence (AI).**

I. INTRODUCTION

The term vectorizer itself conveys a primary meaning "vector" which means it has some directional things. We can define vectors with its close relative term Matrix. Similarly vector also has a size. As we are considering textual data for a machine learning model, we converting text to numbers with this vectorizers. The process of converting textual data to numerical data is called as "Vectorizing". Fitting of a ML model for a textual data is achieved with the help of vectorizers. The textual data may include a paragraph or a group of words or some letters to categorize. The below image represents the concept of vectorization.

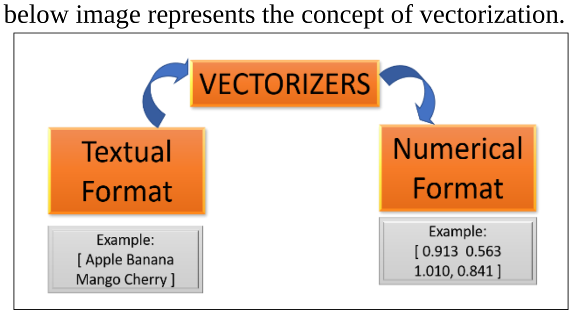

Fig 1. Concept of Vectorizers.

Types of Vectorizers:
Vectorizers[1-8] are primary method to convert textual data to numerical data. When we have a different type of data, we deal with different types of vectorizers.

1. Count Vectorizers:
The input textual data has a structure of sentences, we can process with Count Vectorizers. In general count vectorizers are able to differentiate vocabulary from the sentence input. Once vectorizer was initialized with count vectorizers module from sci-kit learn, we are fitting the sentence into vectorizers. From the below example we can clearly observe what is

happening with count vectorizers for a sentence structure.

Example sentence: “Hi hello how are you, I am from India. We are using vectorizers to categorize text to numerical.”

After vectorization process, we can observe a numerical array format for the specific sentence. The numerical array is size of One row and 15 columns matrix of class integer with NumPy. The size of the matrix is [1 x 15]. Numerical array is as this format [[2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1]]. The logic behind these count vectorizers are, primarily these Count Vectorizers differentiate unique vocabulary from the input sentence and fixes the size of array. For our input sentence we have 15 unique words. So, the size of output NumPy array is 15. Once vocabulary is fixed, it will compare the frequency of the word with in sentence.

```
sentence = """Hi hello how are you,
           Iam from India. We are using vectorizers
           to cateorize text to numericals."""
```

```
vect_data = cv.fit_transform([sentence])
```

```
array(['are', 'cateorize', 'from', 'hello', 'hi', 'how', 'iam', 'india',
       'numericals', 'text', 'to', 'using', 'vectorizers', 'we', 'you'],
      dtype=object)
```

```
numerical_array
```

```
array([[2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1]])
```

Fig 2. Representation of Count Vectorizers for a Single Sentence.

Count Vectorizers for a paragraph: While we are dealing with a paragraph count vectorizers collects all the vocabulary in all sentences and makes a numerical array with vocabulary length. This will create a multi-dimensional NumPy array. The example of a paragraph is explained. Example paragraph is “I love India”. “I am a python programmer”. “Programming is a fun task”. We got the numerical array as [3 x9] matrix. [[0 0 1 0 1 0 0 0 0][1 0 0 0 0 1 0 1 0][0 1 0 1 0 0 1 0 1]].

```
[37] documents = [
         "I love India",
         "I am a python programmer",
         "Programming is a fun task"
     ]
```

```
Document-Term Matrix:
 [[0 0 1 0 1 0 0 0 0]
 [1 0 0 0 0 1 0 1 0]
 [0 1 0 1 0 0 1 0 1]]
```

Fig 3. Usage of count vectorizers for a paragraph.

2. Tokenizers

The world of textual data which is dealing with Natural Language Processing (NLP) uses tokenizers concept. The tokenizers are very similar to count vectorizers. Tokenizers are available in both Natural Language Toolkit (NLTK) and Keras module. From nltk, tokenizers first convert paragraphs into sentences and then sentences to words to numerical. Tokenizers from keras module arranges all vocabulary in an order through frequency count of a word in total vocabulary. The highest frequency word gets lowest number and vice versa. Here is the example usage of a paragraph “Hi all, how are you. We are discussing about the topic tokenizers. We are considering both the module nltk and keras preprocessing module.” The result is paragraph is divided into sentences and then words with the help of word tokenizer from nltk. The result is ['Hi all, how are you.', 'We are discussing about the topic tokenizers.', 'We are considering both the module nltk and keras preprocessing module.'], after the word tokenizers we need to consider for further evaluation. The below picture shows the details.

```
sent_tokenize(paragraph)

['Hi all, how are you.',
 'We are discussing about the topic tokenizers.',
 'We are considering both the module nltk and keras preprocessing module.']
```

Fig 4. Sentences defined for toekniztion.

The word tokenizer result is ['Hi', 'all', ',' ,'how', 'are', 'you.', 'We', 'are', 'discussing', 'about', 'the', 'topic', 'tokenizers.', 'We', 'are', 'considering', 'both', 'the', 'module', 'nltk', 'and', 'keras', 'preprocessing', 'module.'"']. Keras module handles paragraphs different which is based on word frequency count. Word Index is {'are': 1, 'we': 2, 'the': 3, 'module': 4, 'hi': 5, 'all': 6, 'how': 7, 'you': 8, 'discussing': 9, 'about': 10, 'topic': 11, 'tokenizers': 12, 'considering': 13, 'both': 14, 'nltk': 15, 'and': 16, 'keras': 17, 'preprocessing': 18} and we got a numeric array from NumPy as follows Sequences: [[5, 6, 7, 1, 8], [2, 1, 9, 10, 3, 11, 12], [2, 1, 13, 14, 3, 4, 15, 16, 17, 18, 4]]

Normalization is used to train a ML model for better model training. We'll discuss in next section of this survey.

3. TF-IDF

The term TF-IDF is elaborated as Term Frequency Inverse Document Frequency. Term frequency is similar to the concept of count vectorizers, it calculates the frequency of a word in a document or a paragraph. After that document frequency is calculated as inverse of a document which are having the word and total number of documents. The final product of term frequency and Invers of Document frequency is known as TF-IDF value. Mathematical representation of TF is shown below

$$TF = \frac{\text{WORD FREQUENCY}}{\text{TOTAL NUMBER OF WORDS FOR ENTIRE PARAGRAPH}}$$

$$IDF = \log\left[\frac{\text{DOCUMNET CONTAINING WORD}}{\text{TOTAL NUMBER OF PARAGRAPHS/DOCUMENTS}}\right]$$

$$TF\text{-}IDF = TF * IDF$$

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
tfidf = TfidfVectorizer()
```

```
transformed = tfidf.fit_transform(texts)
```

Fig 5. TF-IDF Vectorizers.

Here some other researches may use inverse as logarithmic value. Here is the example for tokenizers. We got numerical array as below array([0.47952794, 0.47952794, 0.47952794, 0.47952794, 0.28321692, 0., 0., 0., 0., 0.,0.,0., 0., 0., 0.,0., 0., 0.]).

4. One Hot Encoders

One hot encoder's works very similar to count vectorizers. Let we have a sentence of words, firstly vocabulary count is calculated and the according to vocabulary length NumPy array column length is calculated and based on the frequency of word with respect to vocabulary count. Consider a paragraph "This is data science domain. We are dealing with data. Machine learning is related to data science. Deep learning is a sub domain of Machine learning". Now the numerical representation of this paragraph is as [[0. 1. 0. 0.] [0. 0. 0. 1.] [0. 0. 1. 0.] [1. 0. 0. 0.]].

```
from sklearn.preprocessing import OneHotEncoder
```

```
One-Hot Encoded Vectors:
 [[0. 1. 0. 0.]
 [0. 0. 0. 1.]
 [0. 0. 1. 0.]
 [1. 0. 0. 0.]]

Vocabulary:
 [array(['Deep learning is a sub domain of machine learning',
       'This is data science domain',
       'machine learning is related to data science',
       'we are dealing with data'], dtype='<U49')]
```

Fig 6. One Hot Encoded Values Text to Numerical

II. CORE TECHNOLOGY-APPLICATIONS WITH ML, DL AND TRANSFORMER TYPE MODELS

1. Machine Learning Model

Machine Learning model fitting is the next step after converting textual data into a numeric array. Classification models are suitable for classification analysis and Naïve Bayes models are mostly preferred while handling textual data. One of the fines bayes model we can use for easy understanding is "Bernoulli Naïve Bayes". We can import BernoulliNB() model from sci-kit learn module, ensemble models. Here is the model selection and import process.

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import BernoulliNB
```

Fig 7. Selecting BernoulliNB ML model.

Once we imported Bernoulli NB model to our workspace, we can fit our count vectorizer results through train-test-split to ML model. Once model is fit with specific data, we can predict the final result. We consider a data frame with few rows and columns. Consider a single column with textual data. So here, we convert the specific text column using Count Vectorizer to a numeric column and fit the specific data frame to ML model. Below image represents the usage of count vectorizer and fitting of ML model for a textual data. In major Naïve Bayes models are suitable for textual data handling part. Assume a variable "x" is a data frame which contains some rows of textual data as shown in figure.

```
x = np.array(data["CONTENT"])
y = np.array(data["CLASS"])
```

```
x[:2]
```

```
array(['Huh, anyway check out this you[tube] channel: kobyoshi02',
       "Hey guys check out my new channel and our first vid THIS IS US THE  MONKEYS!!! I'm the monkey in the white shirt,ple
ase leave a like comment  and please subscribe!!!!"],
      dtype=object)
```

Fig. 8 Variable "x" has some textual data.

Initialize a Count Vectorizer as cv and fit the "x" to Count Vectorizer with specific data frame. Once this step is completed, we can use train_test_split module to split the data into training and testing data. Divided data is fitted with BernoulliNB() model and predict the results.

```
cv = CountVectorizer()
```

```
x = cv.fit_transform(x)
```

Fig 9. Initializing Count Vectorizer for a Textual Data

```
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.2, random_state=42)
```

```
model = BernoulliNB()
model.fit(xtrain, ytrain)
print(model.score(xtest, ytest))
```

```
0.9857142857142858
```

Fig 10. Fitting BernoulliNB Model with Count Vectorizer Data.

From the above figures we can estimate a training accuracy score as 98 percent for a specific textual data. Usage of Count Vectorizers is not appropriate for too large textual data frames due to larger computational time and memory is required to handle all the data.

2. Deep Learning Model with Natural Language Processing

Deep Learning is a subset of Machine Learning. Neural Network type architecture is present in DL models. It has a input layer and some hidden layers and final output layer. Let us consider a Data frame which has a column of textual data. The next step to process this textual data is tokenizing. Tokenizer is available from keras module. Once we initiated Tokenizer and applied the textual data to the tokenizer the results are available in numerical format.

Example textual data is a list of sentences:

```
[1] sentence_list = ["i didnt feel humiliated",
                     "i can go from feeling so hopeless to so damned hopeful just from being around someone who cares and is awake",
                     "im grabbing a minute to post i feel greedy wrong",
                     "i am ever feeling nostalgic about the fireplace i will know that it is still on the property",
                     "i am feeling grouchy",
                     "ive been feeling a little burdened lately wasnt sure why that was",
                     "ive been taking or milligrams or times recommended amount and ive fallen asleep a lot faster but i also feel like so funny",
                     "i feel as confused about life as a teenager or as jaded as a year old man"
                     ]
```

Fig 11. Sentence list definition.

Once sentence list is prepared, tokenizing the list converts these sentences to a numerical format. Here is the numerical format of above sentences. We used padding concept to achieve uniform length numerical array.

```
[21] padded_sequences

array([[ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
         0,  0,  0,  1, 18,  3, 19],
       [ 0,  0,  1, 20, 21,  9,  4,  6, 22, 10,  6, 23, 24, 25,  9, 26,
        27, 28, 29, 30, 11, 12, 31],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 32, 33,  2,
        34, 10, 35,  1,  3, 36, 37],
       [ 0,  0,  0,  0,  0,  1, 13, 38,  4, 39, 14, 15, 40,  1, 41, 42,
        16, 43, 12, 44, 45, 15, 46],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
         0,  0,  0,  1, 13,  4, 47],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  7, 17,  4,  2, 48,
        49, 50, 51, 52, 53, 16, 54],
       [ 7, 17, 55,  8, 56,  8, 57, 58, 59, 11,  7, 60, 61,  2, 62, 63,
        64,  1, 65,  3, 66,  6, 67],
       [ 0,  0,  0,  0,  0,  0,  1,  3,  5, 68, 14, 69,  5,  2, 70,  8,
         5, 71,  5,  2, 72, 73, 74]], dtype=int32)
```

Fig 12. Tokenized sentences.

Apply this tokenized[9-15] data to a deep leaning model with good number of epochs, so that model training accuracy will be increased. Below image describes a Sequential Model which has one input layer, tokenized data is fed into this layer, has two hidden layers with "relu" activation and one output layer. This model is compiled and executed with 10 number epochs for simple data. Here is the result image shown below.

```
[ ] # Define the model
    model = Sequential()
    model.add(Embedding(input_dim=len(tokenizer.word_index) + 1,
                        output_dim=128, input_length=max_length))
    model.add(Flatten())
    model.add(Dense(units=128, activation="relu"))
    model.add(Dense(units=len(one_hot_labels[0]), activation="softmax"))

    model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
    model.fit(xtrain, ytrain, epochs=10, batch_size=32, validation_data=(xtest, ytest))

Epoch 1/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step - accuracy: 0.5000 - loss: 1.3815 - val_accuracy: 0.5000 - val_loss: 1.3936
Epoch 2/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 353ms/step - accuracy: 0.8333 - loss: 1.1709 - val_accuracy: 0.0000e+00 - val_loss: 1.4010
Epoch 3/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 96ms/step - accuracy: 0.8333 - loss: 1.0107 - val_accuracy: 0.0000e+00 - val_loss: 1.4050
Epoch 4/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step - accuracy: 0.8333 - loss: 0.8638 - val_accuracy: 0.0000e+00 - val_loss: 1.4089
Epoch 5/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 140ms/step - accuracy: 0.8333 - loss: 0.7294 - val_accuracy: 0.0000e+00 - val_loss: 1.4091
Epoch 6/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step - accuracy: 0.8333 - loss: 0.6077 - val_accuracy: 0.0000e+00 - val_loss: 1.4041
Epoch 7/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step - accuracy: 0.8333 - loss: 0.4998 - val_accuracy: 0.0000e+00 - val_loss: 1.3911
Epoch 8/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 134ms/step - accuracy: 0.8333 - loss: 0.4056 - val_accuracy: 0.0000e+00 - val_loss: 1.3679
Epoch 9/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - accuracy: 1.0000 - loss: 0.3248 - val_accuracy: 0.0000e+00 - val_loss: 1.3341
Epoch 10/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step - accuracy: 1.0000 - loss: 0.2555 - val_accuracy: 0.0000e+00 - val_loss: 1.2915
<keras.src.callbacks.history.History at 0x797b1d64a6b0>
```

Fig 13. Deep Learning Model with Tokenizer Concept and Model Fitting.

3.Transformer Model with Tokenizers concept.

Transformer models are extended version of deep learning, which are able to generate new textual data based on the trained data. When new text prompt is given to this model, it processes the data and produces the text generated from trained data. A simple transformer model is considered below for this survey analysis. Consider a simple 'BERT' transformer model with tokenizer. BERT is a Bidirectional Representations from Transformers. This model uses tokenizers from hugging face platform.

```
[23] sample_text = "This is a sample text for survey paper for vectorizers and tokenizers"

[24] tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

     tokenizer_config.json: 100%  48.0/48.0 [00:00<00:00, 1.18kB/s]
     vocab.txt: 100%  232k/232k [00:00<00:00, 3.93MB/s]
     tokenizer.json: 100%  466k/466k [00:00<00:00, 4.09MB/s]
     config.json: 100%  570/570 [00:00<00:00, 7.76kB/s]

     input_layer = tokenizer(sample_text, return_tensors="pt")
```

Fig 14. Transformer model from Hugging face platforms.

III. RESULTS

As we discussed different types of vectorizers and their application across models, choosing the exact vectorizer or tokenizers purely depends on the use case of a machine learning model. If we are about to generate series of sentences, tokenizers would be a good choice in some cases. For word-based gaming models word vectorizers or count vectorizers are good choice. Below image represent comparison of count vectorizer and tokenizers.

First image represents usage of count vectorizers and the second image shows usage of tokenizers in for the same text documents.

```
[31] # Sample text data
     documents = [
         "I love my country",
         "India is my country",
         "India has highest population among all countries",
         "I love cricket",
     ]
```

Fig 15. Sample text documents for survey analysis.

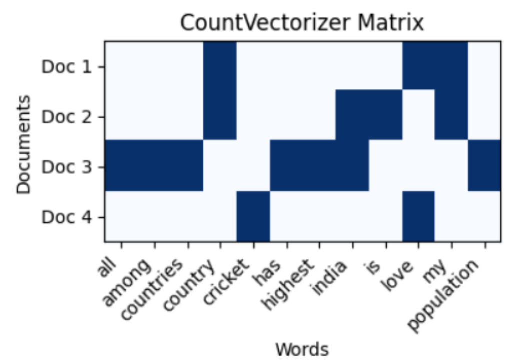

Fig 16. Count vectorizer plots for text for four documents

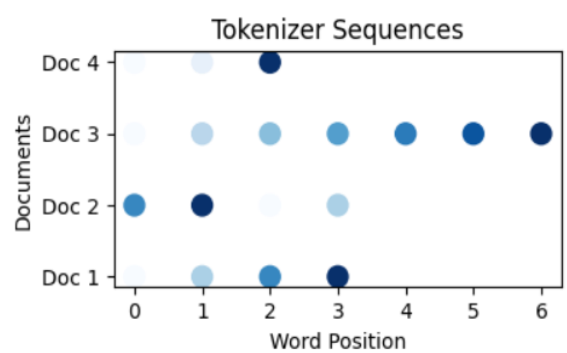

Fig 17. Tokenizers from Keras module in for same four text documents.

CONCLUSION

This survey paper discussed about effective usage of vectorizers from small scale ML models to large language models (Transformers). Different types of text to numeric conversation helpful tin training machine learning models.

REFERENCES

[1] Machine Learning in python via scikit learn official from https://scikit-learn.org/stable/

[2] Transformer models from hugging face platform, https://huggingface.co/

[3] Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. Advances in Neural Information Processing Systems, pp. 1–9

[4] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2. (Publisher link).

[5] The pandas development team, ‘pandas-dev/pandas: Pandas’. Zenodo, Apr. 10, 2024. doi: 10.5281/zenodo.10957263.

[6] Buitinck, L. et al., 2013. API design for machine learning software: experiences from the scikit-learn project. In ECML PKDD Workshop: Languages for Data Mining and Machine Learning. pp. 108–122.

[7] Chollet F, others. Keras [Internet]. GitHub; 2015. Available from: https://github.com/fchollet/keras

[8] Nazzal, Fayez & Abualrob, Ahmad & Husni, Ahmad. (2022). Classification of Item's category using the CountVectorizer, The Linear Support Vector Machine Algorithm and Python's sklearn module. 10.13140/RG.2.2.24423.32169.

[9] Yang, Jinbiao. (2024). Rethinking tokenization: Crafting better tokenizers for large language models. International Journal of Chinese Linguistics. 11. 94-109. 10.1075/ijchl.00023.yan.

[10] (2024). Large language models (LLMs): survey, technical frameworks, and future challenges. Artificial Intelligence Review. 57. 10.1007/s10462-024-10888-y.

[11] Yang, Jingfeng & Jin, Hongye & Tang, Ruixiang & Han, Xiaotian & Feng, Qizhang & Jiang, Haoming & Zhong, Shaochen & Yin, Bing & Hu, Xia. (2024). Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond. ACM Transactions on Knowledge Discovery from Data. 18. 10.1145/3649506.

[12] Ananthaswamy, Anil. (2023). In AI, is bigger always better?. Nature. 615. 202-205. 10.1038/d41586-023-00641-w.

[13] Raza, Muhammad & Meghji, Areej & Mahoto, Naeem & Al Reshan, Mana & Abosaq, Hamad & Sulaiman, Adel & Shaikh, Asadullah. (2024). Reading Between the Lines: Machine Learning Ensemble and Deep Learning for Implied Threat Detection in Textual Data. International Journal of Computational Intelligence Systems. 17. 10.1007/s44196-024-00580-y.

[14] Ulah, Arif & Khan, Sundas & Mohd Nawi, Nazri. (2022). Review on sentiment analysis for text classification. Multimedia Tools and Applications. 82. 10.1007/s11042-022-14112-3.

[15] Zhang, Tianyi & Ladhak, Faisal & Durmus, Esin & Liang, Percy & McKeown, Kathleen & Hashimoto, Tatsunori. (2024). Benchmarking Large Language Models for News Summarization. Transactions of the Association for Computational Linguistics. 12. 39-57. 10.1162/tacl_a_00632.