Combining KNN and Sentence Transformers for Article Categorization

Christopher Brokenshire
College of William and Mary
Email: cmbrokenshire@wm.edu

Matthew Berthoud
College of William and Mary
Email: mwberthoud@wm.edu

Lekha Reddy
College of William and Mary
Email: Lkreddy@wm.edu

***Abstract*—This paper presents a novel approach to categorizing news articles using a combination of K-Nearest Neighbors (KNN) classification and Sentence Transformers. The primary aim of this study is to develop a method that efficiently organizes large volumes of news articles based on their titles, thereby enhancing the accessibility of information and improving user experience. We leverage the SentenceTransformers framework for vectorizing article titles into numerical embeddings, which are then classified using a custom KNN algorithm designed for tensor data. This approach capitalizes on the principle that articles with similar titles often share the same category. Our results demonstrate a significant accuracy score, indicating the efficacy of our method in categorizing news articles. The simplicity of our model, compared to more complex NLP systems, offers a computationally efficient alternative while maintaining a high level of accuracy. The paper also discusses the limitations of our approach, particularly in domain invariance, and suggests potential areas for future improvement. This study contributes to the field of natural language processing by demonstrating the application of traditional machine learning techniques in addressing modern-day challenges in information categorization.**

I. Introduction

News articles can pertain to a vast array of topics. Grouping articles based on the category that best aligns with their content is an efficient way to organize the immense amount of information. A few benefits of categorizing articles is that each article is more likely to reach its target audience, and each user is better able to find appropriate information. However, it takes substantial effort to sort articles based on their entire contents. With this in mind, our project aims to optimize the process by implementing a KNN classification algorithm to categorize articles based on their titles.

Our algorithm will capitalize on the idea that articles with similar titles are likely to fall into the same category. This justifies the utilization of a KNN approach, which is designed around the foundational principle that similar data points in a feature space will have similar target values. In an effort to boost accuracy, we plan to quantify the similarity between two titles, or the distance between two data points, using the cosine-similarity calculated by the SentenceTransformers python framework.

We will begin by discussing the concept and workings of a Sentence Transformer. Afterwards, we will detail the dataset we selected for building and testing the model. Following this, we will delve into the construction and tuning of the custom KNN classification model we created, later presenting the results of our testing. Finally, we will analyze the model's successes and failures in achieving its intended goals and discuss the reasons behind the results.

II. Applied Approaches

A. Sentence Transformers

The SentenceTransformers framework vectorizes string data, according to how the model specified by an argument is trained. For our purposes we used the all-MiniLM-L6-v2 which is a MiniLM model trained over a billion training pairs. The strings are vectorized based on Siamese BERT-Networks, a methodology explained in Reimers et. al ., 2018 [1]. The vectorization takes into account semantic meanings, word sizes, and more, based on how it is trained. We could have trained our own data, but we felt this would introduce too much of our own bias into the data, and it would be hard to clearly define our own degree of textual similarity with which to validate the encodings. The time complexity of encodings increases parabolically with the length of the string, but since we were encoding titles and not full articles, this was hardly a relevant consideration. Example code from the Sentence-Transformers documentation for a simple set of encodings is below.

```
from sentence_transformers import
    SentenceTransformer
model = SentenceTransformer('all-MiniLM-L6-v2')

#Our sentences we like to encode
sentences = ['This framework generates embeddings
    for each input sentence',
        'Sentences are passed as a list of string.',
        'The quick brown fox jumps over the lazy dog
    .']

#Sentences are encoded by calling model.encode()
embeddings = model.encode(sentences)

#Print the embeddings
for sentence, embedding in zip(sentences, embeddings
    ):
    print    ("Sentence:", sentence)
    print    ("Embedding:", embedding)
    print    ("")

#https://www.sbert.net/#usage
```

Listing 1. Encoding Example: https://www.sbert.net/usage

The `embedding` variable in this snippet is a numpy array of length 384. There are therefore 384 dimensions to these encodings.

After encoding strings in this way, we were able to calculate the "distance" between them as necessary for the KNN method. Our manner of calculating distance was cosine similarity. This is a standard technique for comparing textual data in Natural Language Processing. We demonstrate the resultant similarity values with some simple examples below, using the same libraries and `model` variable as Listing 1.

```
def cos_sim_test(s1, s2):
    emb1     = model.encode(s1)
    emb2     = model.encode(s2)
    cos_sim     = float(util.cos_sim(emb1, emb2))
    return     cos_sim

tests = [
        ["cat", "dog"],
        ["cat", "Cat"],
        ["Hark upon the Gale", "Once upon a time"],
        ["Hark upon the Gale", "Professor Ye Gao
    rocks"],
        ["Hark upon the Gale", "Professor Gao rocks"
    ]
]

for pair in tests:
    print    (f"{format(cos_sim_test(pair[0], pair
    [1]),'.6f')}    {pair}")
```

Listing 2. SentenceTransformer Testing

This code outputs the following values similarity values:

```
0.66063768  ['cat', 'dog']
0.99999988  ['cat', 'Cat']
0.24494889  ['Hark upon the Gale', 'Once
upon a time']
0.07204005  ['Hark upon the Gale',
'Professor Ye Gao rocks']
0.01792862  ['Hark upon the Gale',
'Professor Gao rocks']
```

This demonstrates that factors like word length, number of words, shared words, semantic content, and even (very subtly) capitalization all play a part in the 384 vectorized fields of the encoded data. [2]

B. Dataset

Our dataset consists of 210,294 titles of news articles published between 2012 and 2022 that were gathered from HuffPost and uploaded to Kaggle [3]. The following 6 features are provided for each item in the dataset: the category of the article, its headline, the contributing authors, a link to the original article, a short description of its contents, and the date of publication. There are 42 categories into which the articles are sorted. The largest 5 categories include Politics, Wellness, Entertainment, Travel, and Style & Beauty, which contain 43%, precisely 90,623 data points, of the total data.

This News Category Dataset best aligns with our project's intentions as it provides a sufficient quantity of articles that have already been categorized by their most relevant topic and are easily downloadable. This enhances our workflow as we can efficiently use python to pre-process the data by isolating the headline and category features. The process of calculating the accuracy of our model's predictions is also optimized as the dataset has already established which category each data point best corresponds with.

The potential for bias stems from the heavy over-representation of data from 2012 to May 2018. 95% of the headlines in this dataset are dated between 2012 and May 2018, meaning that it only includes 10,000 headlines from May 2018 to 2022. The significance of this uneven distribution is that it may not predict future articles' categories in the most modern fashion. If recent years have demonstrated a shift in the general consensus of which category a certain article should belong in, it is less likely to be reflected in our model's predictions.

C. Model

Our project adapts the K-Nearest Neighbors (KNN) algorithm for text data, specifically article titles. We start by extracting the titles and their corresponding categories into separate arrays. Using a sentence transformer, each title is then converted from a string to an embedded tensor, enabling us to compare titles numerically. The challenge was that standard KNN is designed for numerical data, not tensors. To address this, we developed a custom KNN subclass that can handle tensor data. This involved initializing our datasets in the subclass differently. Thankfully, KNN does have a feature where we can input our own custom metric as a parameter when initializing the class. Our metric function finds the cosine similarity between two embeddings. Each embedding is attributed to the category of the title string it came from. Our KNN algorithm classifies title strings by classifying their embedding. In other words, it finds the majority category from the k closest embeddings. Below I have attached the custom KNN class the team developed.

```
def cosine_similarity_1(x, y):
    cos_sim = util.cos_sim(x, y)
    return cos_sim

class KNNStringClassifier(KNeighborsClassifier):
    def _fit(self, X, y):
        self._fit_X = X
        self._fit_y = y
        self._y = y
        return self

    def predict(self, X):
        predictions = []
        for x in X:
            #compute similarities to all points in
    self._fit_X
            #convert each tensor to a scalar and
    store it in similarities
            similarities = [self.metric(x, x_train).
    item() for x_train in self._fit_X]

            #find indices of k most similar
    neighbors
            nn_indices = np.argsort(similarities)[-
    self.n_neighbors:]

            #determine the most common class among
    nearest neighbors
```

```
            neighbors_classes = [self._fit_y[idx] for idx in nn_indices]
            most_common_class = Counter(neighbors_classes).most_common(1)[0][0]
            predictions.append(most_common_class)

        return predictions


    def evaluate(self, X, y_true):
        #implement custom evaluation logic for classification
        y_pred = self.predict(X)
        accuracy = np.mean([pred == true for pred, true in zip(y_pred, y_true)])
        return accuracy
```

Listing 3. KNN Class

D. Validation

The next step for preparing our model is tuning it. Thankfully, in the case of KNN tuning usually just means finding the correct k value. We tuned our model by having a training set of 10,000 titles and a validation set 1,000. This will give us the opportunity to see which k value tends to lead to the best accuracy score. Due to the nature of the strings and how much they can vary, I decided to first see how accuracy scores varied globally. So we ran a loop that calculated the accuracy score from k from 1 to 100 by increments of 5 as you can see in the chart below:

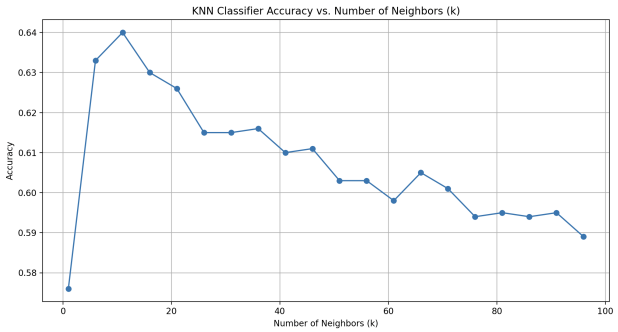

Fig. 1. Accuracy Score for K from 1 to 100

As you can see in the graph, the highest accuracy score is concentrated between k =1 to k=20. So let us see the graph of the k value in that region. See Fig.2

From illustration we see we hit a maximum at k = 9 with an accuracy score of 64.2

III. Evaluation

A. Results

It's time to test the model. We chose the test set to be a 1000 datapoint subset of the dataset that the model had not seen during training and validation. When we tested our model on this subset we got the following accuracy score: 0.619. Is this good?

On the surface one may think that the model is under performing but the reality is, for the simplicity of the model,

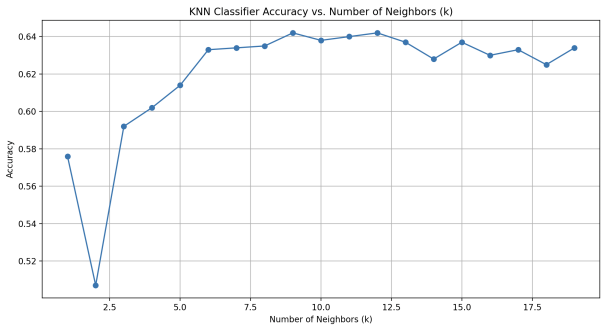

Fig. 2. Accuracy Score for K from 1 to 20

it's actually quite a remarkable score. For context, there are 42 categories which means that our model is very far from random. Additionally, on Kaggle the ranked models for this dataset have an accuracy score of a little over 0.7. These models used advanced NLP models like BERT to achieve their results. It's reassuring to know that such a simple concept is 0.08 away from the state of the art.

IV. Discussion

The clear advantage of our model is its simplicity. It leverages straightforward KNN regression to address a complex modern-day NLP issue, such as article classification. Yet, in the grand scheme of ML, while simplicity is important, accuracy is what truly matters. So, the question really is: can we improve the model?

It's first smart to consider what our model is doing in the context of this problem. The answer to this question is heavily reliant on how the sentence transformer determines similarity. For our project, we used one of the standard pretrained sentence transformer models: all-MiniLM-L6-v2. As discussed earlier, this model determines sentence similarity based on several attributes, such as contextual word meanings, word order and grammatical structure, and sentence length. How the model determines sentence similarity is heavily dependent on its training. From what my team understood about the model we selected, these are the factors that determines sentence similarity.

What are the implications of our metric? It's clear that the metric we have chosen, while seemingly effective, results in our model being minimally domain invariant. To explain what I mean, consider three articles: 'Biden's Favorite Place to Go in Tahiti,' 'Biden Goes to Tahiti to Visit the President,' and 'Tourist's Favorite Places to Visit in the Bahamas.' When we compare these sentences to each other, the cosine similarity between the first and second sentence is 0.8, while the cosine similarity between the first and third is 0.2. The significance of these results is that the first and last sentences are clearly about travel, while the second is about news. Although this is an edge case, it highlights a significant issue with our metric: it's not comparing the abstract similarities, its not comparing the generalized attributes that determine the similarity between all news or travel articles.

So, the question now is: why is it working so well? My team believes that it has something to do with topic recurrence. Newspapers aim to sell as many copies as possible. As a result, if they write an article about Biden and it sells well, they will continue to write articles about him. The same goes for topics like travel or parenting; articles such as 'Why Is My Teen So Moody?' or 'Best Places to Go in Italy' will have many variations because they are popular. Therefore, the reason why our model is classifying articles correctly might be because there is an article discussing a variation of the same subject in the training set. Our high accuracy score has much more to do with article writers not straying from what makes them money, and less to do with having captured the abstract idea of what it means for an article to be about healthy living.

My team deduced that for us to improve the model, we must build a similarity metric that compares the similarity between sentence objectives. What information are these sentences trying to convey in the abstract sense. Unfortunately, to do so we would have to train a sentence transformer model which is out of the scope of this project.

I must say, there is an advantage to our current model that we have not yet discussed. Due to the repetitive nature of articles and how our KNN model categorizes them, it's evident that the larger the training set, the higher the classification accuracy. This makes sense because the more training articles we have, the higher the probability that each test article will find a variation of itself in the training set. So, why don't we run the algorithm with a training set of 180,000 and a test set of 18,000? The reason is that KNN is computationally intensive, especially when using a sentence transformer as our similarity metric. Nevertheless, I tried training the model with 70,000 data points and testing it with 7,000. When my team ran the model, we actually achieved an accuracy score of 0.72, which is comparable to state-of-the-art methods. However, it took our computers two days to perform this single computation.I reasoned that if it was too computationally expensive for us to document it extensively with such a large datasets (try other k values), then its most likely not applicable for real world use, especially once we understood why it was doing so well after all.

V. Conclusion

In this project, we embarked on an ambitious journey to categorize news articles efficiently using a combination of K-Nearest Neighbors (KNN) classification and Sentence Transformers. The primary objective was to leverage the power of a simple machine learning algorithm to categorize a vast array of articles based on their titles.

Our approach involved utilizing Sentence Transformers to convert article titles into numerical embeddings, which were then classified using a custom KNN algorithm designed to handle tensor data. The model's simplicity was a significant advantage, allowing us to apply a classic machine learning technique to a modern NLP challenge. We found that titles with similar embeddings often corresponded to articles within the same category, underscoring the effectiveness of our method.

Through rigorous testing and validation, our model achieved a noteworthy accuracy score of 0.619, which was somewhat surprising given the straightforward nature of our research. However, our project also uncovered limitations, particularly in the domain invariance of our model. The metric we used for determining similarity, while effective, tended to focus more on topical similarity rather than abstract thematic resemblance. This limitation could be resolved in future work involving a more nuanced metric or exploring other machine learning models that might capture the essence of article content more accurately.

In conclusion, our project has demonstrated the potential of combining KNN with Sentence Transformers for categorizing news articles. While there is room for improvement, particularly regarding abstract similarity detection, the success of our model in achieving considerable accuracy with a relatively straightforward approach is encouraging. Future work could explore more advanced models or refined metrics to further enhance the accuracy and applicability of this method in the ever-evolving field of NLP.

References

[1] Reimers, Nils, and Iryna Gurevych. "Sentence-bert: Sentence embeddings using Siamese Bert-Networks." *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing* (EMNLP-IJCNLP), 2019, https://doi.org/10.18653/v1/d19-1410.

[2] "SentenceTransformers Documentation — Sentence-Transformers documentation" www.sbert.net/. Accessed 7 Dec. 2023.

[3] Misra, Rishabh. "News Category Dataset." arXiv preprint arXiv:2209.11429 (2022), https://www.kaggle.com/datasets/rmisra/news-category-dataset. Accessed 7 Dec. 2023.