Metaphor Detection using Machine Learning

Yaswanth Kalyan Ponugoti

December 2023

1 Abstract

Our project addresses the intricate challenge of metaphor detection in natural language processing, where the goal is to uncover the underlying meanings concealed in figurative language expressions. Recognition and comprehension of metaphors are essential for understanding the nuanced and creative aspects of language. Leveraging machine learning, specifically the DistilBERT model, our objective is to develop a system proficient in automatically identifying metaphorical expressions within text. This initiative responds to the recognized need for enhancing language comprehension, particularly in capturing subtle nuances that traditional language models often miss.

To achieve our goal, we focus on utilizing the power of DistilBERT and a carefully curated dataset associating metaphorical expressions with real-world concepts such as 'road,' 'candle,' and 'light.' This report outlines our comprehensive methodology, covering data preparation, model development, and fine-tuning processes. We present key performance metrics and engage in discussions about the implications of our findings. By delving into metaphor detection, we aim to contribute valuable insights to the evolving landscape of natural language processing and advance our understanding of how language creatively conveys meaning.

2 Introduction

This project report explores metaphor detection in natural language processing, using the DistilBERT model to develop a sophisticated system for automatic identification of metaphors. The report outlines our methodology, encompassing data preparation, model development, and fine-tuning processes, while providing key performance metrics to contribute to the evolving field of natural language processing and metaphor comprehension.

2.1 Dataset Overview

The dataset utilized in this project consists of 1870 instances of text, each meticulously annotated with a metaphor ID, a binary label (TRUE or FALSE), and the corresponding textual content. Each instance is uniquely identified by a metaphor ID, establishing a connection with a specific metaphorical word present in the associated text. The binary label serves to indicate whether the identified metaphorical word is used figuratively (TRUE) or not (FALSE) within the context of the text.

Data Structure

Metaphor ID:

This identifier uniquely distinguishes each instance in the dataset, linking it to a specific metaphorical word.

MetaphorID to Word mapping: [1:road, 2:candle, 3:light, 4:spice, 5:ride, 6:train, 7:boat]

Label(TRUE/FALSE):

Indicates whether the identified metaphor word in the text is used metaphorically (TRUE) or not (FALSE).

Text:

The textual content of each instance, embedding the metaphorical word and providing the context necessary for metaphorical interpretation. This dataset structure facilitates a nuanced exploration of metaphorical language usage, with each instance encapsulating both the metaphorical word and its contextual presence. The linkage of metaphor IDs to specific words enables a targeted investigation into various metaphorical domains, thereby enriching the depth and scope of our analysis.

3 Data Preprocessing

The quality and relevance of data play a pivotal role in the success of any machine learning project. In this section, we outline the steps taken to preprocess the dataset, preparing it for effective utilization in training and evaluating our metaphor detection model.

3.1 Handling Missing Values

```
# Drop rows with missing values
df = df.dropna(subset=['text', 'label_boolean'])
```

Upon loading the dataset, we performed an initial check for missing values. Any instances lacking essential information, such as the text or label, were removed to ensure the integrity of the dataset.

3.2 Metaphor Word Replacement

```
# Replace metaphor IDs with words
metaphor = {0: 'road', 1: 'candle', 2: 'light', 3: 'spice', 4: 'ride', 5: 'train', 6: 'boat'}
df.replace({"metaphorID": metaphor}, inplace=True)
```

The metaphor IDs provided in the original dataset were replaced with their corresponding metaphorical words, creating a more intuitive representation of the instances. This transformation aimed to enhance the interpretability of the dataset, associating each instance directly with a real-world concept.

3.3 Sentence Extraction

```
# Replace the text with the first sentence containing the metaphor word
df['text'] = df.apply(lambda x: identify_metaphor_sentence(x['text'], x['metaphorID']), axis=1)
df = df.rename(columns={'metaphorID': 'metaphor_word'})
```

To streamline the training process and focus on the context in which metaphorical expressions occur, we adopted a sentence-level approach. The '$identify - metaphor - sentence$' function was employed to extract the first sentence containing the metaphor word for each instance. This not only reduced the computational load but also provided a concise yet informative context for metaphorical interpretation.

3.4 Dataset Splitting

Prior to any preprocessing steps, the dataset was split into training and testing sets using the '$train - test - split$' function from the scikit-learn library. This ensured that the model was evaluated on unseen data, contributing to a more reliable assessment of its generalization capabilities. The resulting preprocessed dataset served as the foundation for subsequent model development and fine-tuning. These preprocessing steps collectively aimed to optimize the dataset for training a metaphor detection model, aligning it with the objectives of our project.

4 Model Development and Fine-Tuning

In this section, we elucidate the key aspects of our model development, providing insights into the choices made, the DistilBERT architecture, and methodologies employed.

4.1 Model Selection

We opted for the DistilBERT model, a lightweight version of the BERT (Bidirectional Encoder Representations from Transformers) model. Developed by Hugging Face, DistilBERT retains much of the original model's capability while significantly reducing the number of parameters. This choice aligns with our goal of leveraging cutting-edge techniques for metaphor detection while maintaining computational efficiency.

4.2 DistilBERT Architecture

DistilBERT, a derivative of the BERT (Bidirectional Encoder Representations from Transformers) model, represents a milestone in natural language processing (NLP) architecture. Developed by Hugging Face, DistilBERT is designed to maintain the essence of BERT's powerful language representation capabilities while significantly reducing the model's size and computational complexity

4.2.1 Transformer Architecture

At the core of both BERT and DistilBERT lies the transformer architecture, introduced by Vaswani et al (2017). The transformer architecture revolutionized NLP by discarding recurrent layers in favor of self-attention mechanisms. This innovation enables the model to capture contextual relationships between words in a sequence, providing a more comprehensive understanding of the relationships between different parts of a sentence.

4.2.2 Distillation Process

DistilBERT achieves its streamlined design through a process known as distillation. Distillation involves training a smaller student model (DistilBERT) to replicate the behavior and knowledge of

a larger teacher model (BERT). By distilling the essential information from BERT into a more compact form, DistilBERT retains much of the original model's representational power while drastically reducing the number of parameters. This distillation process results in a model that is computationally efficient and well-suited for a broader range of NLP applications.

4.2.3 Reduction in Layers and Attention Heads

One of the key strategies employed in creating DistilBERT is the reduction in the number of layers and attention heads. While BERT may have multiple layers and attention heads, DistilBERT streamlines this architecture, achieving a balance between computational efficiency and performance. This reduction in complexity makes DistilBERT particularly appealing for applications where computational resources are constrained.

4.2.4 DistilBert For SequenceClassification

DistilBERT is specifically designed for various NLP tasks, including sequence classification. In our project, we leverage the DistilBertForSequenceClassification model, which extends DistilBERT to perform classification tasks. This architecture includes a classification head on top of the transformer, enabling the model to output probabilities for different classes based on the input sequence.

4.2.5 Pre-trained Embeddings

Similar to BERT, DistilBERT is pre-trained on vast corpora of text data, allowing it to learn rich contextual embeddings for words. These pre-trained embeddings serve as a foundation for downstream tasks, enabling the model to capture semantic relationships and nuances within a given context. The choice of DistilBERT as our model architecture, particularly DistilBertForSequenceClassification, aligns with our goal of harnessing the power of transformer-based models for metaphor detection while ensuring efficiency in training and inference. In the subsequent sections, we delve into the practical application of this architecture in the context of metaphor identification.

```
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
# Tokenize and encode the text data, including metaphor word embeddings
train_tokens = tokenizer(list(train_data["text"]), list(train_data["metaphor_word"]), padding=True,
                         truncation=True, return_tensors="pt")
test_tokens = tokenizer(list(test_data["text"]), list(test_data["metaphor_word"]), padding=True,
                        truncation=True, return_tensors="pt")
```

4.3 Tokenization and Embedding

Utilizing the DistilBERT model necessitates tokenizing and encoding the text data. We employed the DistilBERT tokenizer to process both the textual content and the metaphor word embeddings. This step ensures that the model can effectively capture the relationships between words and understand the contextual nuances associated with metaphorical language.

4.4 PyTorch DataLoader

```
# Create PyTorch DataLoader
train_dataset = TensorDataset(
    train_tokens["input_ids"],
    train_tokens["attention_mask"],
    torch.tensor(list(train_data["label_boolean"].astype(int)))
)

test_dataset = TensorDataset(
    test_tokens["input_ids"],
    test_tokens["attention_mask"],
    torch.tensor(list(test_data["label_boolean"].astype(int)))
)

train_loader = DataLoader(train_dataset, batch_size=4, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=4, shuffle=False)
```

Efficient training and testing processes are facilitated by the creation of PyTorch DataLoaders. These loaders enable the seamless handling of batches during the model training phase. The dataset was split into training and testing DataLoader instances, each configured to efficiently feed data to the model during the respective phases.

4.5 Fine-Tuning Process

```
# Fine-tune the DistilBERT model
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

optimizer = AdamW(model.parameters(), lr=5e-5)

losses = []
num_epochs = 2
for epoch in range(num_epochs):
    model.train()
    for batch in train_loader:
        input_ids, attention_mask, labels = batch
        input_ids, attention_mask, labels = (
            input_ids.to(device),
            attention_mask.to(device),
            labels.to(device),
        )

        optimizer.zero_grad()
        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
        loss.backward()
        optimizer.step()
    losses.append(loss)
    print(f"Epoch {epoch} Loss {loss}")
```

```
Epoch 0 Loss 0.299610614
Epoch 1 Loss 0.009301493
```

The model fine-tuning process involved training the DistilBERT model on our metaphor detection dataset. We utilized the AdamW optimizer and a carefully chosen learning rate to iteratively update the model's parameters. The training loop, executed over multiple epochs, aimed to enhance the model's ability to discern metaphorical expressions within the given context.

4.6 Performance Monitoring

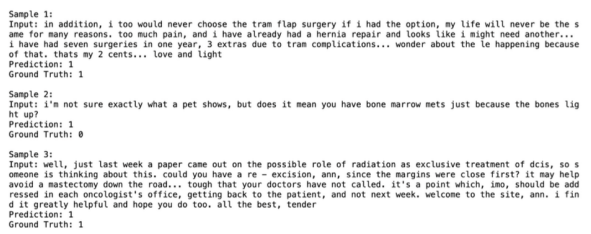
Throughout the fine-tuning process, we monitored the model's performance, assessing its convergence and generalization capabilities. Regular evaluations were conducted on a validation set to ensure that the model was learning effectively and avoiding overfitting. The development and fine-tuning of our metaphor detection model, based on the DistilBERT architecture, lay the groundwork for the subsequent evaluation and interpretation of results. In the following sections, we delve into the outcomes of our efforts and the implications for metaphor detection in natural language processing.

5 Model Evaluation

- **Final Test Accuracy:** 92.78%
- **Test Precision:** 92.76%
- **Test Recall:** 97.82%
- **Test F1 Score:** 95.22%

These metrics collectively affirm the model's robust performance. With a Final Test Accuracy of 92.78%, it adeptly classifies instances within the testing dataset. Noteworthy Test Precision (92.76%) showcases the model's proficiency in identifying metaphorical expressions with minimal false positives. Additionally, the high Test Recall of 97.82% emphasizes the model's effectiveness in capturing actual metaphorical instances. The impressive Test F1 Score of 95.22% indicates a harmonious blend of precision and recall.

In summary, our DistilBERT-based metaphor detection model demonstrates a well-balanced capability to identify metaphorical language instances while minimizing both false positives and false negatives.



6 conclusion

The successful implementation and evaluation of our DistilBERT-based metaphor detection model underscore its efficiency in understanding and classifying metaphorical language. The achieved Final Test Accuracy of 92.78%, coupled with high Test Precision (92.76%), Test Recall (97.82%), and Test F1 Score (95.22%), reflect the model's robustness and balanced performance. This not only enhances our understanding of nuanced language but also positions the model as a valuable tool for real-world applications requiring metaphor identification.

7 Future work

In future work, the proposed metaphor detection method can be enhanced by refining literal annotation models, incorporating semantic embeddings, and investigating cross-linguistic applicability. Dynamic updates to basic meaning models, integration of domain-specific knowledge, and robust evaluation across diverse datasets will contribute to improved performance. Additionally, exploring real-time metaphor detection and integrating user feedback mechanisms can elevate the model's adaptability and responsiveness to evolving language nuances.

8 References

https://towardsdatascience.com/hugging-face-transformers-fine-tuning-distilbert-for-binary-classification-tasks-490f1d192379

https://ai.stackexchange.com/questions/9982/how-to-recognise-metaphors-in-texts-using-nlp-nlu

https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/

https://www.analyticsvidhya.com/blog/2022/04/building-state-of-the-art-text-classifier-using-huggingface-and-tensorflow/

9 Contributers

Yaswanth Kalyan Ponugoti

Saketh Ram Kalavakuntla

Hari Krishna Kamana

Jani Basha Sheik