CommonLit - Evaluate Student Summaries

Automatically assess summaries written by students in grades 3-12

Introduction

The dataset has approximately 24,000 summaries crafted by students spanning grades 3 to 12. These summaries contains a diverse array of subjects and styles. For each summary, it has undergone an assessment resulting in two distinct scores: one for its content and another for its wording. The primary aim of this competition is to anticipate both content and wording scores for summaries that pertain to unfamiliar topics.

Here is a concise overview of the dataset files and their respective fields:

1. summaries_train.csv: This file contains summaries from the training set and includes the following fields:

- student_id: Identifies the student writer.

- prompt_id: Associates the summary with a specific prompt, linking to the prompt file.

- text: Presents the complete text of the student's summary.

- content: Indicates the content score for the summary (first target).

- wording: Specifies the wording score for the summary (second target).

2. summaries_test.csv: Summaries from the test set, featuring the same fields as summaries_train.csv, except for content and wording.

3. prompts_train.csv: This file furnishes prompts for the training set and consists of these fields:

- prompt_id: Serves as a unique identifier, linking prompts to the corresponding summaries in summaries_train.csv.

- prompt_question: Delivers the specific question or prompt that students were tasked with responding to, outlining the primary theme or topic of the required summaries.

- prompt_title: Provides a brief title summarizing the prompt's subject matter.

- prompt_text: Contains the full text of the prompt, affording additional context and details regarding what students were expected to summarize.

4. prompts_test.csv: Holds prompts for the test set, with fields identical to those in prompts_train.csv. Note that prompts in the train/public test/private test splits are distinct.

5. sample_submission.csv: This file is a template for submissions, structured correctly. Please refer to the Evaluation page for specific details.

Note: Our task at first will be to understanding the dataset's structure and the essence of content and wording scores is necessary, as the competition's objective is to build models which can accurately predicting these scores for fresh, unseen summaries. Secondly, Use the provided information, training data, and potential feature engineering to construct a robust predictive model for content and wording scores.

Here's a high-level outline of the pipeline:

1. **Data Loading and Preprocessing**:

- Load the summaries_train.csv and prompts_train.csv files using pandas.

- Merge the data based on the prompt_id to connect summaries with their corresponding prompts.

- Perform any necessary preprocessing, such as text cleaning and feature extraction.

2**. Feature Engineering:**

- Generate relevant features from the text data that might aid in predicting content and wording scores. This could involve techniques like TF-IDF, word embeddings, or other domain-specific features.

3. **Model Building:**

- Split the dataset into training and validation sets.

- Choose a machine learning algorithm (e.g., XGBoost) to build separate models for predicting content and wording scores.

- Train the models on the training set.

4**. Model Evaluation:**

- Evaluate the trained models on the validation set using appropriate evaluation metrics (e.g., Mean Squared Error for regression tasks).

5. **Model Tuning:**

- Tune hyperparameters of the models to achieve better performance.

6. **Generate Predictions:**

- Load the summaries_test.csv and prompts_test.csv files for the test set.

- Preprocess and transform the test data using the same steps as the training data.

- Use the trained models to predict content and wording scores for the test set summaries.

7**. Create Submission File:**

- Organize the predicted scores and any required IDs into a submission format.

- Save the submission as a CSV file for submission in the competition.

Here's a simplified code snippet to give you an idea of how the pipeline might look:

```
#python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error

# Load and merge data
summaries_train = pd.read_csv('summaries_train.csv')
prompts_train = pd.read_csv('prompts_train.csv')
data = pd.merge(summaries_train, prompts_train, on='prompt_id')

# Feature engineering
tfidf = TfidfVectorizer(max_features=1000)  # Adjust max_features as needed
X = tfidf.fit_transform(data['text'])

# Split data
X_train, X_val, y_content_train, y_content_val, y_wording_train, y_wording_val =
train_test_split(
    X, data['content'], data['wording'], test_size=0.2, random_state=42
)

# Build and train models
model_content = XGBRegressor()
model_content.fit(X_train, y_content_train)

model_wording = XGBRegressor()
model_wording.fit(X_train, y_wording_train)

# Evaluate models
y_content_pred = model_content.predict(X_val)
y_wording_pred = model_wording.predict(X_val)

mse_content = mean_squared_error(y_content_val, y_content_pred)
mse_wording = mean_squared_error(y_wording_val, y_wording_pred)

print(f"Content MSE: {mse_content}")
print(f"Wording MSE: {mse_wording}")

# Load test data
summaries_test = pd.read_csv('summaries_test.csv')
prompts_test = pd.read_csv('prompts_test.csv')
test_data = pd.merge(summaries_test, prompts_test, on='prompt_id')

# Preprocess test data and make predictions
X_test = tfidf.transform(test_data['text'])
content_predictions = model_content.predict(X_test)
wording_predictions = model_wording.predict(X_test)

# Create submission file
submission = pd.DataFrame({
    'prompt_id': test_data['prompt_id'],
    'content': content_predictions,
    'wording': wording_predictions
})
submission.to_csv('submission.csv', index=False)
```

Remember that this is a basic example, and you might need to adjust the preprocessing steps, feature engineering, and model selection to achieve the best results. Additionally, consider hyperparameter tuning and exploring more advanced techniques for better model performance.

What Types of Models you can use here?

Modern language models like Transformers and BERT have proven to be highly effective in a variety of natural language processing (NLP) tasks, including regression tasks like predicting scores. These models have revolutionized the field of NLP due to their ability to capture contextual information and relationships within text data. Here are some model options that can work well for predicting content and wording scores in your dataset:

1. BERT (Bidirectional Encoder Representations from Transformers):

BERT is a pre-trained transformer model that excels in understanding the context of words in a sentence. You can fine-tune a pre-trained BERT model for your regression task. It requires tokenizing your text data and incorporating the scores as regression targets.

2. RoBERTa (A Robustly Optimized BERT Pretraining Approach):

RoBERTa is an optimization of BERT that further improves performance by using larger batch sizes and more training data. Like BERT, it can be fine-tuned for regression tasks.

3. DistilBERT:

DistilBERT is a distilled version of BERT that retains much of BERT's performance while being more memory-efficient and faster. This can be advantageous when working with limited computational resources.

4. XLNet:

XLNet is another transformer-based model that has shown strong performance on various NLP tasks. It captures dependencies in both directions and has the ability to model context more effectively.

5. GPT (Generative Pre-trained Transformer):

While GPT models are primarily designed for text generation, they can also be used for regression tasks. You can fine-tune a pre-trained GPT model by providing the content and wording scores as targets.

6. T5 (Text-to-Text Transfer Transformer):

T5 frames all NLP tasks as a text-to-text problem, where both the input and output are treated as text sequences. It's versatile and can be fine-tuned for regression tasks as well.

7. ALBERT (A Lite BERT for Self-supervised Learning of Language Representations):

ALBERT is another variation of BERT that focuses on reducing the parameter count while maintaining strong performance. It can be used similarly to BERT for regression tasks.

When using these models for regression, you'll need to fine-tune them on your training data with the content and wording scores as your target variables. Most of these models can be accessed through the Hugging Face Transformers library, which provides pre-trained models and tools for fine-tuning. Keep in mind that the choice of model depends on factors such as the available computational resources, dataset size, and desired performance. Experimenting with different models and architectures, along with proper hyperparameter tuning, will help you find the best approach for your specific task.

Types Of FE:

Textual Feature Engineering (Level: Easy)

- Number of words in text
- Number of unique words in text
- Number of characters in text (including whitespaces)
- Number of stop words in text
- Number of punctuations in text
- Number of upper-case words in text
- Number of title-case words in text
- Average word length in text

Textual Feature Engineering (Level: Medium)

- Number of paragraphs in a text
- Number of sentences in text
- Number of contractions (can't, won't, don't, haven't, etc.) in text
- Text Polarity
- Text Subjectivity

Textual Feature Engineering (Level: Medium)

- Number of dialogues and narratives
- Mean word length for dialogues and narratives
- Count of part-of-speech (POS) tagging in text