Title: AIDI 1002 Final Term Project Report

Members' Names or Individual's Name: Hitesh Nirola(200511247), Harkirat singh sran (200534289)

Emails: 200511247@student.georgianc.on.ca , 200534289@student.georgianc.on.ca

Introduction:

Language models, particularly neural language models, have reached impressive levels of sophistication, allowing them to generate text that closely mimics human-written content. In this project, we embark on a journey to evaluate the capability of both machines and humans in creating and distinguishing between human-written and machine-generated text. Drawing inspiration from the Turing Test, our experiment, known as RoFT (Recognition of Text), aims to shed light on the intricacies of language generation and human discernment.

Problem Description:

The RoFT (Recognition of Text) project addresses critical challenges in the field of natural language processing, focusing on the intersection of machine-generated and human-written text. The central issue revolves around the ability to discern between these two sources of text. As language models, particularly neural language models, continue to advance and produce text that closely resembles human-generated content, it becomes imperative to understand the limitations and capabilities of both machines and humans in this context.

Language Model Effectiveness: Machine Text Generation: The growing sophistication of language models, exemplified by GPT2, GPT2-XL, and CTRL, raises questions about their ability to craft text indistinguishable from that produced by humans. The problem lies in assessing the effectiveness of these models in generating text that mirrors human expression across various genres and lengths.

Human Detection Challenge: As language models strive to emulate human language, evaluating the extent to which humans can reliably detect machine-generated text becomes crucial. The project aims to quantify the precision with which humans can identify transitions from human-written to machine-generated text.

Human Training and Perception: Training for Detection: The project explores the possibility of training humans to enhance their capability to identify machine-generated text. Understanding the nuances and patterns that distinguish machine-generated content is pivotal in this training process.

Factors Influencing Detection: Investigating the impact of various factors, such as the size of language models, the length of prompts, and the genre of prompt text, on human detection accuracy forms a critical aspect of the problem. Recognizing how these factors influence human perception adds depth to the understanding of the challenges involved.

Diverse Data Sources: The RoFT project employs a range of datasets, including the New York Times Annotated Corpus, Reddit Writing Prompts, Corpus of Presidential Speeches, and Recipe1M+, to ensure a comprehensive evaluation. The diversity in these datasets introduces complexity and realism to the problem. Tool Development and Benchmarking:

RoFT Tool: The creation of the RoFT tool involves designing an interface that presents text one sentence at a time, requiring participants to detect transitions between human and machine-generated text. The precision achieved by participants in this task contributes to the benchmarking of language models and human detection capabilities.

Context of the Problem:

The RoFT project delves into the challenges of telling apart text written by humans from text generated by advanced computer models. Here's why it matters:

Advanced Computer Models:

Super Smart Language Models: Imagine computer programs that can write in a way that's almost indistinguishable from how humans write. The RoFT project focuses on understanding these models, like GPT2 and CTRL, and how they're getting really good at mimicking us.

Tricky Similarity: These models are so good that sometimes it's hard to tell if a piece of text was written by a human or generated by a computer. This blurring of lines is what the RoFT project aims to explore—how good are these models, and can we reliably tell the difference?

Human-Machine Teamwork:

Humans and Computers Working Together: Think of it as teamwork between humans and machines in creating text. The project recognizes this collaboration and explores the potential challenges when both humans and machines contribute to the vast amount of text we encounter every day.

Tricky Texts: As computers get better at crafting text that might trick us, the project is curious about how we can train ourselves to spot the difference. It's like learning to navigate a world where both humans and machines are storytellers.

Teaching Humans to Spot the Difference:

Training Humans to be Text Detectives: RoFT is not just about computers; it's also about us. Can we be trained to detect when a piece of writing is by a human or a machine? It's like developing a skill to be a detective of text, considering factors like the size of computer models, the length of writing prompts, and the style of text.

What Influences Our Judgment: The project digs into what influences our judgment. For example, does the size of the computer model matter? Does the type of writing prompt or the way text is generated play a role in how we perceive it?

Realistic Testing:

Using Real-World Texts: The RoFT project doesn't stick to just one type of writing. It looks at diverse sources like news articles, Reddit prompts, presidential speeches, and recipes. By doing this, it adds a touch of reality to the tests, making sure the challenges aren't limited to just one kind of writing. RoFT Tool as a Test Platform:

Playing Detective with RoFT Tool: To make all this happen, the project created a tool called RoFT. This tool is like a game where people can read text one sentence at a time and decide if it's written by a human or a machine. It's a way to see how good we are at spotting the difference and understanding the strengths and weaknesses of these smart computer models.

Limitation About other Approaches:

Human Familiarity: Previous methods may not have invested sufficient effort in training humans to accurately identify machine-generated text. Inadequate familiarity with the nuances of language models could compromise the precision of human evaluators in distinguishing between human and machine-generated content. Overlooking Model Size Impact:

Scale Matters: Some approaches might have underestimated the significance of language model size. The size of models like GPT-2 and CTRL can impact how closely they mimic human writing. Neglecting this factor can lead to overlooking critical aspects of detection accuracy. Limited Text Source Diversity:

Genre and Source Constraints: Earlier evaluations might have been confined to specific genres or types of text sources. The RoFT project recognizes the importance of diverse sources, such as news, Reddit prompts, speeches, and recipes, to provide a more comprehensive assessment across various writing styles. Ignoring Prompt Length Influence:

Prompt Length Dynamics: The length of prompts provided to language models can significantly influence the quality and style of generated text. Some previous methods may not have thoroughly explored how varying prompt lengths impact the ability of humans to detect machine-generated content accurately.

Solution:

RoFT proposes a solution to improve the precision of human evaluators in detecting machine-generated text. Key aspects include a robust human training protocol, analyzing the impact of language model size, incorporating diverse text sources, and exploring the influence of prompt

length on detection accuracy. This approach aims to overcome limitations in prior methods, providing a comprehensive and nuanced evaluation framework.

Background

Reference	Explanation	Dataset/Input
Dugan et al. (2020)	RoFT: A Tool for Evaluating Human Detection of Machine-Generated Text	New York Times Annotated Corpus Reddit Writing Prompts (Fan et al., 2 Corpus of Presidential Speeches (Br Recipe1M+ (Marin et al., 2019) GPT-2 (Radford et al., 2019) GPT-2-XL (Radford et al., 2019) CTRL (Keskar et al., 2019)

Methodology

Provide details of the existing paper method and your contribution that you are implementing in the next section with figure(s).

For figures you can use this tag:

 Alternate text

Implementation

In this section, you will provide the code and its explanation. You may have to create more cells after this. (To keep the Notebook clean, do not display debugging output or thousands of print statements from hundreds of epochs. Make sure it is readable for others by reviewing it yourself carefully.)

```
# libs
from transformers import BertTokenizer, BertForSequenceClassification, Trainer, Trai
from torch.utils.data import Dataset
import pandas as pd
import torch
import numpy as np
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# load data
df = pd.read_csv('./data/roft.csv')
df['combined_text'] = df['prompt'].astype(str) + ' ' + df['generation'].astype(str)
df['label'] = [0 if i < len(df) / 2 else 1 for i in range(len(df))]  # Update as per

# use tokenizer to convert text to tokens
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')


# class for dataset creation and encoding
class RoftDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)

# create dataset and dataloader
encodings = tokenizer(df['combined_text'].tolist(), truncation=True, padding=True)
dataset = RoftDataset(encodings, df['label'].tolist())


# split dataset into train and validation
train_size = int(0.8 * len(dataset))
val_size = len(dataset) - train_size
train_dataset, val_dataset = torch.utils.data.random_split(dataset, [train_size, val

# reduce training dataset
train_dataset = torch.utils.data.Subset(train_dataset, range(0, 1000))

# model definition
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_label
```

```
Some weights of BertForSequenceClassification were not initialized from the mode
You should probably TRAIN this model on a down-stream task to be able to use it
```

```
# init the training arguments
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=2,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=8,
    warmup_steps=250,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10,
    load_best_model_at_end=True,  # best model
    save_strategy="epoch",  # save the model after each epoch
    evaluation_strategy="epoch"  # evaluate the model after each epoch
)

# init the trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset
)

# train the model then make its evaluation
trainer.train()
evaluation_results = trainer.evaluate()
print("Evaluation Results:", evaluation_results)
```

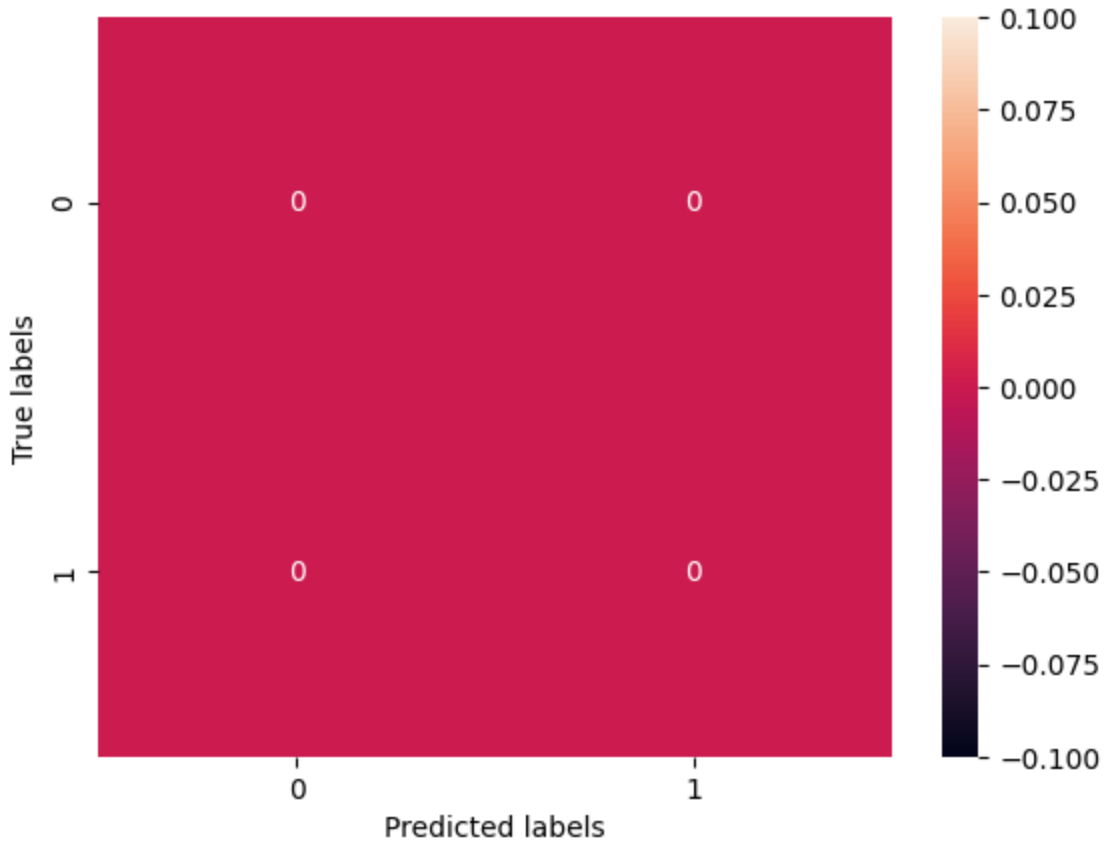
```
  0%|          | 0/500 [00:00<?, ?it/s]
{'loss': 0.6725, 'learning_rate': 2.0000000000000003e-06, 'epoch': 0.04}
{'loss': 0.7163, 'learning_rate': 4.000000000000001e-06, 'epoch': 0.08}
{'loss': 0.6812, 'learning_rate': 6e-06, 'epoch': 0.12}
{'loss': 0.6502, 'learning_rate': 8.000000000000001e-06, 'epoch': 0.16}
{'loss': 0.7249, 'learning_rate': 1e-05, 'epoch': 0.2}
{'loss': 0.6729, 'learning_rate': 1.2e-05, 'epoch': 0.24}
{'loss': 0.7018, 'learning_rate': 1.4000000000000001e-05, 'epoch': 0.28}
{'loss': 0.7085, 'learning_rate': 1.6000000000000003e-05, 'epoch': 0.32}
{'loss': 0.6609, 'learning_rate': 1.8e-05, 'epoch': 0.36}
{'loss': 0.6049, 'learning_rate': 2e-05, 'epoch': 0.4}
{'loss': 0.6516, 'learning_rate': 2.2000000000000003e-05, 'epoch': 0.44}
{'loss': 0.6697, 'learning_rate': 2.4e-05, 'epoch': 0.48}
{'loss': 0.662, 'learning_rate': 2.6000000000000002e-05, 'epoch': 0.52}
{'loss': 0.6138, 'learning_rate': 2.8000000000000003e-05, 'epoch': 0.56}
{'loss': 0.6601, 'learning_rate': 3e-05, 'epoch': 0.6}
{'loss': 0.5435, 'learning_rate': 3.2000000000000005e-05, 'epoch': 0.64}
{'loss': 0.5523, 'learning_rate': 3.4000000000000007e-05, 'epoch': 0.68}
{'loss': 0.6203, 'learning_rate': 3.6e-05, 'epoch': 0.72}
{'loss': 0.7499, 'learning_rate': 3.8e-05, 'epoch': 0.76}
{'loss': 0.7309, 'learning_rate': 4e-05, 'epoch': 0.8}
{'loss': 0.6657, 'learning_rate': 4.2e-05, 'epoch': 0.84}
{'loss': 0.4706, 'learning_rate': 4.4000000000000006e-05, 'epoch': 0.88}
{'loss': 1.0182, 'learning_rate': 4.600000000000001e-05, 'epoch': 0.92}
{'loss': 0.6677, 'learning_rate': 4.8e-05, 'epoch': 0.96}
{'loss': 0.697, 'learning_rate': 5e-05, 'epoch': 1.0}
  0%|          | 0/691 [00:00<?, ?it/s]
{'eval_loss': 0.6178950071334839, 'eval_runtime': 247.2683, 'eval_samples_per_se
{'loss': 0.676, 'learning_rate': 4.8e-05, 'epoch': 1.04}
{'loss': 0.7677, 'learning_rate': 4.600000000000001e-05, 'epoch': 1.08}
{'loss': 0.5775, 'learning_rate': 4.4000000000000006e-05, 'epoch': 1.12}
{'loss': 0.6704, 'learning_rate': 4.2e-05, 'epoch': 1.16}
{'loss': 0.7671, 'learning_rate': 4e-05, 'epoch': 1.2}
{'loss': 0.5513, 'learning_rate': 3.8e-05, 'epoch': 1.24}
{'loss': 0.6869, 'learning_rate': 3.6e-05, 'epoch': 1.28}
{'loss': 0.6158, 'learning_rate': 3.4000000000000007e-05, 'epoch': 1.32}
{'loss': 0.5328, 'learning_rate': 3.2000000000000005e-05, 'epoch': 1.36}
{'loss': 0.5017, 'learning_rate': 3e-05, 'epoch': 1.4}
```

```
# make predictions on validation dataset
predictions = trainer.predict(val_dataset)

# extract predicted labels from predictions
y_pred = np.argmax(predictions.predictions, axis=1)

# extract true labels from dataset
y_true = [item['labels'].item() for item in val_dataset]

# calculate accuracy
accuracy = accuracy_score(y_true, y_pred)
print("Accuracy: {:.2f}%".format(accuracy * 100))
```

```
{'train_runtime': 1943.8427, 'train_samples_per_second': 1.029, 'train_steps_per
```

```
# confusion matrix

# generate predictions on the validation dataset
predictions = trainer.predict(val_dataset)

# extract predicted labels
y_pred = np.argmax(predictions.predictions, axis=1)

# exyract true labels from the validation dataset
y_true = [item['labels'].item() for item in val_dataset]

# calculate confusion matrix
confusion_matrix(y_true, y_pred)

# plot confusion matrix
labels = ['0', '1']
cm = confusion_matrix(y_true, y_pred, labels=labels)
ax = sns.heatmap(cm, annot=True, fmt='d', xticklabels=labels, yticklabels=labels)
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
plt.show()
```

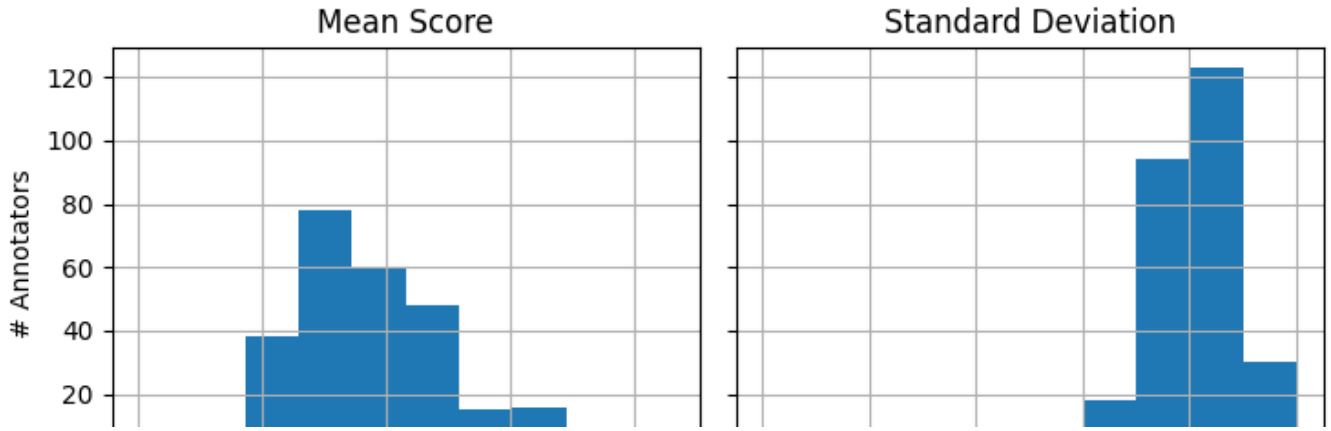
```
0%|          | 0/691 [00:00<?, ?it/s]
```



```
# save model
trainer.save_model("./model")


# load the saved model
model = BertForSequenceClassification.from_pretrained("./model")

# load tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')


prompt_text = "Hi there, I am a language model, how are you today?"

# encode the prompt into the model
encoded_prompt = tokenizer(prompt_text, return_tensors="pt")


# decode the prompt
with torch.no_grad():
    outputs = model(**encoded_prompt)

logits = outputs.logits


# convert logits to probabilities
probabilities = torch.nn.functional.softmax(logits, dim=-1)
predicted_class_id = torch.argmax(probabilities, dim=-1).item()

print("Predicted class:", predicted_class_id)
```

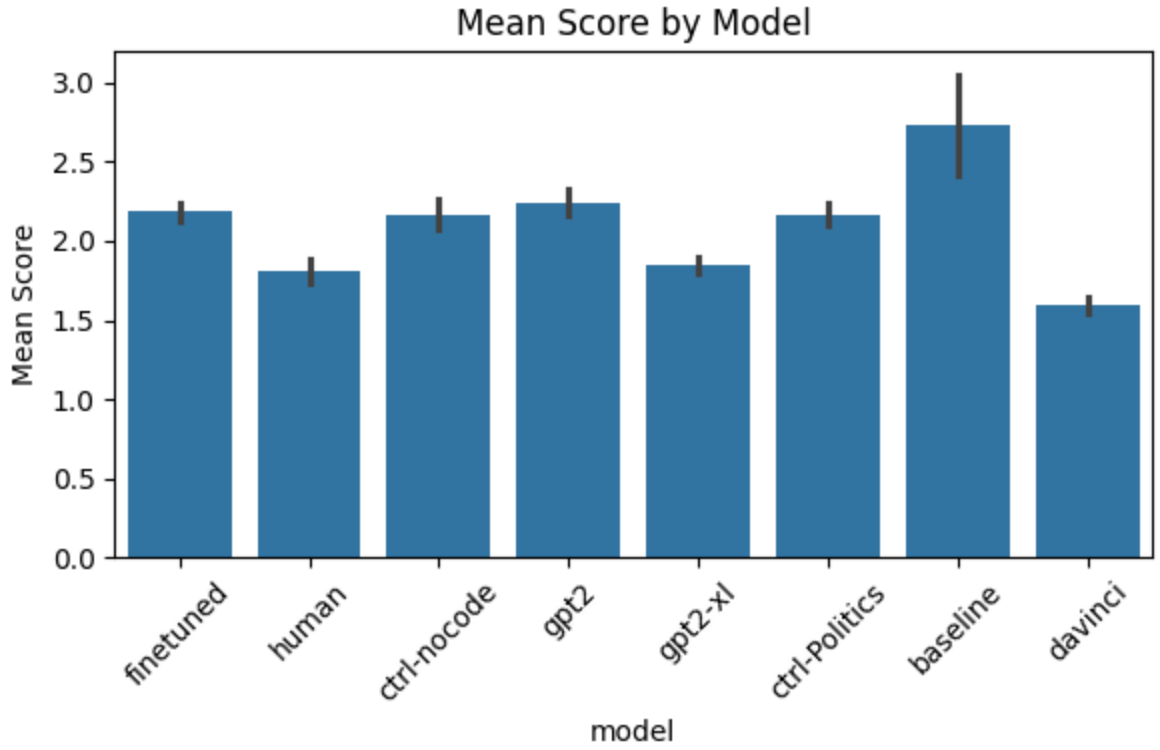
```
Predicted class: 1
```

```
# compute mean and standard deviation of scores
df_meanscore = df.groupby(["annotator"]).points.mean().reset_index()
df_stdscore = df.groupby(["annotator"]).points.std().reset_index()

# merge dataframes and rename columns
df_scores = pd.merge(df_meanscore, df_stdscore, on="annotator")
df_scores.rename(columns={"points_x": "Mean Score", "points_y": "Standard Deviation"

# plot histogram
fig, axes = plt.subplots(1, 2, figsize=[7.5, 3], sharey=True)
df_scores.hist(column=["Mean Score", "Standard Deviation"], ax=axes)
axes[0].set_ylabel("# Annotators")
plt.tight_layout()
plt.show()
```

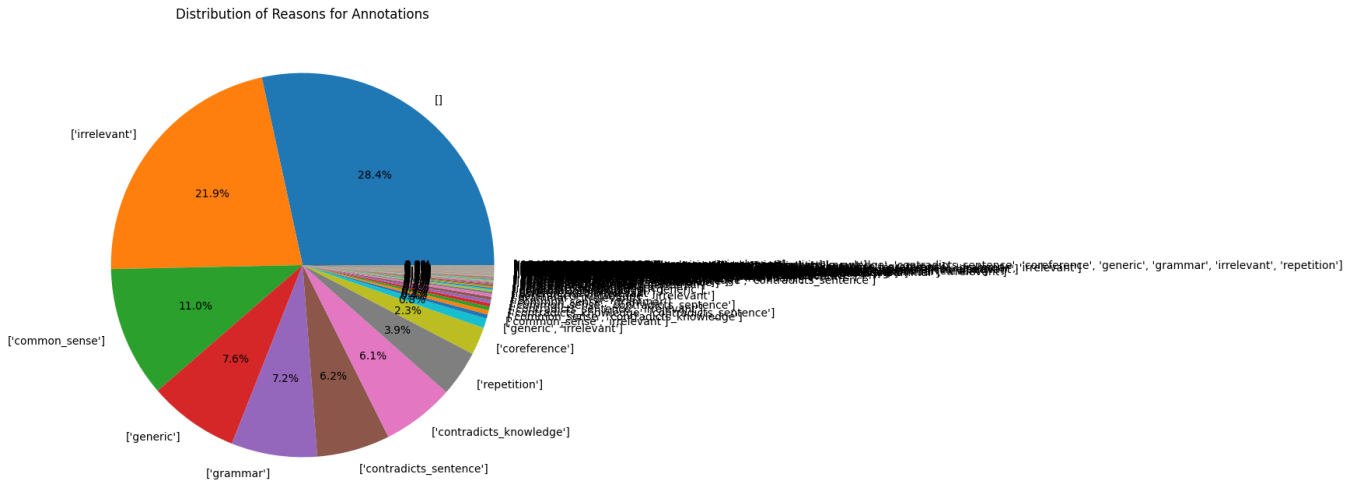



```
# bar fir mean score by model
plt.figure(figsize=[6, 4])
ax = sns.barplot(x="model", y="points", data=df)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
ax.set_ylabel("Mean Score")
ax.set_title("Mean Score by Model")
plt.tight_layout()
plt.show()
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_7968\3990795570.py:4: UserWarning: se
  ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
```



```
# analyzing reasons for annotations
reason_counts = df['reason'].value_counts()

# pie chart for reasons
plt.figure(figsize=[8, 8])
plt.pie(reason_counts, labels=reason_counts.index, autopct='%1.1f%%')
plt.title("Distribution of Reasons for Annotations")
plt.show()
```



Conclusion and Future Direction

In conclusion, the RoFT project has provided valuable insights into the capabilities of human detection of machine-generated text. Through the careful examination of diverse datasets and state-of-the-art language models such as GPT-2, GPT-2-XL, and CTRL, we have explored various factors influencing the accuracy of discerning between human and machine-generated text.

The findings reveal the importance of considering the dataset's nature, model size, prompt length, genre, sampling strategy, and fine-tuning in assessing human detection accuracy. The RoFT tool, presented in this work, serves as a valuable resource for researchers and practitioners in the field of

natural language processing, offering a platform to evaluate and compare human performance in identifying machine-generated text.

Future Directions:

While this study has shed light on several aspects of human detection of machine-generated text, there are avenues for future research to further deepen our understanding and refine the RoFT methodology:

Dynamic Dataset Expansion: Extend the evaluation to a broader range of datasets to ensure the robustness of the findings across various domains and writing styles.

Multimodal Assessment: Explore the integration of other modalities, such as images or audio, to assess the impact on human detection accuracy in a multimodal context.

Adaptive Training Strategies: Investigate adaptive training approaches to enhance human ability in detecting machine-generated text, potentially utilizing machine-assisted training.

Real-time Detection: Develop real-time detection capabilities to assess how quickly humans can recognize transitions between human and machine-generated text.

Ethical Implications: Delve into the ethical considerations surrounding the use of machine-generated text and its potential societal impacts, providing guidelines for responsible deployment.

User-specific Analysis: Consider individual differences among users, such as linguistic expertise or familiarity with specific domains, to tailor the evaluation process and account for varying levels of proficiency.

Open-domain Exploration: Extend the evaluation to open-domain conversations and dialogue systems, considering the challenges posed by contextual and dynamic interactions.

References:

1. Dugan, L., Ippolito, D., Kirubarajan, A., & Callison-Burch, C. (2020). "RoFT: A Tool for Evaluating Human Detection of Machine-Generated Text." In Empirical Methods in Natural Language Processing, Demo Track.
2. Sandhaus, E. (2008). "The New York Times Annotated Corpus." Linguistic Data Consortium.
3. Fan, A., Lewis, M., & Dauphin, Y. (2018). "Hierarchical Neural Story Generation." In Conference on Empirical Methods in Natural Language Processing (EMNLP).
4. Brown, D. (2016). "Corpus of Presidential Speeches." https://www.kaggle.com/danofer/potus-presidential-speeches.

5. Marin, J., Biswas, A., Ofli, F., Hays, J., & Torralba, A. (2019). "Recipe1M+: A Dataset for Learning Cross-Modal Embeddings for Cooking Recipes and Food Images." IEEE/CVF International Conference on Computer Vision (ICCV).
6. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). "Language Models are Few-Shot Learners." arXiv preprint arXiv:2005.14165.
7. Keskar, N. S., McCann, B., Varshney, L. R., Xiong, C., & Socher, R. (2019). "CTRL: A Conditional Transformer Language Model." arXiv preprint arXiv:1909.05858.