Between artificial and human intelligence-6178

End project - Gon Sarrabia (ID - 207044959)

In this project i aims to examine how language models (LLMs) behave in relation to cognitive biases, with a specific focus on determining whether they display the Conjunction Fallacy in their responses to prompts.

Imports And Constants

```
!pip install -q transformers
!pip install -q -U bitsandbytes
!pip install -q -U git+https://github.com/huggingface/peft.git
!pip install -q -U git+https://github.com/huggingface/accelerate.git

import torch
from scipy.stats import ttest_ind, norm, chisquare
from transformers import AutoTokenizer, AutoModelForCausalLM, BitsAndBytesConfig, pipeline
import random
import numpy as np

import accelerate
import bitsandbytes
import peft
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 119.8/119.8 MB 8.1 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 302.6/302.6 kB 2.4 MB/s eta 0:00:00
  Building wheel for peft (pyproject.toml) ... done
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
  Building wheel for accelerate (pyproject.toml) ... done
```

```
NUM_OF_SCENARIOS=100
BANCHMARK_PATH = "/content/Banchmarks.txt"
BNBN_CONFIG = BitsAndBytesConfig(
    load_in_4bit= True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16
    )
DEVICE = "cuda:0" if torch.cuda.is_available() else "cpu"
```

Cognitive Bais: - Conjunction Fallacy

The Conjunction Fallacy is a cognitive bias where individuals mistakenly believe that the simultaneous occurrence of two events (A and B) is more likely than the occurrence of one of the events individually. This fallacy reflects a misunderstanding of how probabilities combine.

Mathematical Expression

Formally, the probability that both events A and B occur, denoted as $P(A \cap B)$, should always be less than or equal to the probability of either event occurring by itself, expressed as: $P(A \cap B) \leq P(A)$ and $P(A \cap B) \leq P(B)$ This inequality highlights the irrationality of the conjunction fallacy, showing that joint probabilities cannot exceed individual probabilities.

Benchmark:

The Benchmark is designed to create and manage scenarios to test the Conjunction Fallacy. In the Benchmark class by creating profiles with characteristics such as names, traits, and professions the main function generates scenarios inspired by the "Linda problem", originally formulated by Amos Tversky and Daniel Kahneman[A] to study the conjunction fallacy.

Given the following:

- $P(A)$ - Probability of employees who are [Profession].
- $P(B)$ - Probability of employees interested in [Hobby].

- $P(A \cap B)$ -Probability of employees who are both [Profession] and interested in [Hobby].

Each benchmark scenario is structured as follows:

[name] is [trait]. He holds a degree in [major] and enjoys [hobby] in his free time. During college, he was involved in [extra_curricular].True or False: It is more probable that [name] just works as a [profession] than that he works as a [profession] and also [secondary_profession]."

That means thet each scenario present specific probabilities for two attributes and their combined occurrence and ask the question:

"**True or False: It is more probable for:** $P(A \cap B)$**>P(**X**)? or** $P(A \cap B)$**<P(**X**)?**"

These crafted questions challenge the model to evaluate whether the joint occurrence is more likely than one of the individual attributes.

```
class Benchmark:
  def __init__(self):
   self.prompts = None
   self.true_answers = None

  def banchmark_profiles(self):
        names = ["Alex", "Jordan", "Taylor", "Morgan", "Jamie", "Sam", "Casey", "Riley", "Drew", "Robin"]
        ages = ["25", "30", "35", "40", "45", "50"]
        traits = ["creative", "analytical", "outgoing", "dedicated", "thoughtful", "energetic"]
        fields = ["biology", "computer science", "marketing", "engineering", "literature", "physics", "urban planning",
                  "psychology"]
        hobbies = ["painting", "programming", "public speaking", "cycling", "running", "gardening", "cooking",
                   "digital art"]
        extra_activities = ["member of a book club", "volunteer firefighter", "yoga instructor",
                            "board game enthusiast", "podcast host", "local theater actor", "community organizer",
                            "wildlife rescuer"]
        name = random.choice(names)
        age = random.choice(ages)
        trait = random.choice(traits)
        field = random.choice(fields)
        hobby = random.choice(hobbies)
        extra_activity = random.choice(extra_activities)
        return [name, age, trait, field, hobby, extra_activity]

  def banchmark_getter(self, suffle=False):
    if suffle:
      self.generate_conjunction_fallacy_scenarios()
    return self.prompts, self.true_answers

  def generate_conjunction_fallacy_scenarios(self,num_scenarios=NUM_OF_SCENARIOS):
    # prompt = []
    # answer = []
    self.prompts = []
    self.true_answers = []
    for _ in range(num_scenarios):
        name, age, trait, field, hobby, extra_activity = self.banchmark_profiles()

        profile = f"{name} is {age} years old, {trait}, and interested in {field}. In their free time, they enjoy {hobby}. "
        if random.choice([True, False]):
            question = f"True or False: It is more probable that {name} works in {field} than that {name} works in {field} and is also a
            correct_answer = 'true'
        else:
            question = f"True or False: It is more probable that {name} works in {field} and is also a {extra_activity} than that {name}
            correct_answer = 'false'
        self.prompts.append(profile + question)
        self.true_answers.append(correct_answer)


# benchmark = Benchmark()
# prompts ,true_answers = benchmark.banchmark_getter(True)



def load_lists_from_file(filename):
    local_scope = {}
    with open(filename, 'r') as file:
        content = file.read()
        exec(content, {}, local_scope)

    prompts = local_scope['prompt']
    correct_answers = local_scope['true_answers']
    ICL_prompts_and_answers = local_scope['ICL_prompts_and_answers']
    context = ("These prompts are designed to explore the Conjunction Fallacy, where the combined probability "
               "of two events is mistakenly thought to be more likely than one event alone. Engage with each scenario "
               "to challenge and refine your understanding of probability assessments and decision-making biases.")

    return prompts, correct_answers, ICL_prompts_and_answers, context
prompts, true_answers, ICL_prompts_and_answers, context = load_lists_from_file(BANCHMARK_PATH)
```

```
print("prompts=",prompts)
print("true_answers=",true_answers)
```

```
prompts= ['Morgan is 25 years old, energetic, and interested in engineering. In their free time, they enjoy gardening. True or False
true_answers= ['true', 'true', 'true', 'true', 'false', 'true', 'false', 'false', 'true', 'true', 'false', 'false', 'false', 'true',
```

Model evaluation:

Model Evaluation- **Overview**

The evaluation process is specifically designed to measure how accurately language models handle scenarios involving the Conjunction Fallacy. This type of cognitive bias assessment is crucial for determining the reliability of models in tasks requiring nuanced probabilistic reasoning.

Model Evaluation - **Class and Function**

The **ModelEvaluator** class is responsible for generating predictions and assessing the accuracy and bias of LLm based on the predefined benchmarks.

- **'bias_modifier':** Configures the model evaluator with specific strategies for prompt engineering and in-context learning.
- **'evaluater':** Sets up the model and tokenizer, generates answers for the full benchmark, and evaluates model bias.
- **'generate_logits':** Takes an input sequence, processes it using the model's tokenizer, and gets the model's logits for the last token in the sequence. It returns these logits.
- **'calculate_probabilities':** Takes the logits (for the last token), calculates the probabilities using softmax, and accumulates the probabilities for the terms associated with true and false. It then decides the answer by comparing the sums of these probabilities.
- **generate_full_benchmark_answers:** Generates a list of model answers for the entire set of prompts.
- **'evaluate_model_bias':** Performs a chi-square test to compare between the correct answers and the model answers and determines if there any observed bias is statistically significant.
- **'calculate_average_error:'** Calculates the average squered error for wrong answers based on the differences in probabilities.

```
class ModelEvaluator():
    def __init__(self,statements,true_answers):
        self.statements = statements
        self.true_answers = true_answers
        self.model_answers = None
        self.diff_prob_list = None
        #model's settings
        self.model = None
        self.tokenizer = None
        #de-bais
        self.prompt_engineering = None
        self.ICL_prompts_and_answers = None

    def bias_modifier(self, prompt_engineering, icl):
      self.prompt_engineering =prompt_engineering
      self.ICL_prompts_and_answers = icl

    def evaluater(self,model,tokenizer,enc_or_neg_cog_bias=None ):
        self.model = model
        self.tokenizer = tokenizer
        self.generate_full_benchmark_answers(enc_or_neg_cog_bias)
        return self.evaluate_model_bias()

    def generate_logits(self, input_sequence):
      input_ids = self.tokenizer.encode(input_sequence, return_tensors='pt')
      with torch.no_grad():
          logits = self.model(input_ids=input_ids)[0]
      return logits[0, -1, :]


    def calculate_probabilities_and_decide(self, last_token_logits):
        probabilities = torch.softmax(last_token_logits, dim=0).cpu().numpy()
        trues = 0
        falses = 0
        for token in ['true', 'True', 'false', 'False']:
            token_index = self.tokenizer.convert_tokens_to_ids(token)
            if token in ['true', 'True']:
                trues += probabilities[token_index]
            else:
                falses += probabilities[token_index]
        if trues >= falses:
            answer = 'true'
```

```
        else:
            answer = 'false'
        return answer, pow(trues-falses,2)


    def generate_full_benchmark_answers(self,enc_or_neg_cog_bias):
        answer_list = []
        diff_prob_list = []
        counter = 0
        for statement in self.statements:
            if enc_or_neg_cog_bias == "prompt_engineering":
              statement = self.prompt_engineering + "\n" + statement
            if enc_or_neg_cog_bias == "":
              statement = self.ICL_prompts_and_answers + "\n" + statement
            logits = self.generate_logits(statement)
            answer, porb_diff =self.calculate_probabilities_and_decide(logits)
            answer_list.append(answer)
            diff_prob_list.append(porb_diff)
        self.model_answers = answer_list
        self.diff_prob_list = diff_prob_list


    def evaluate_model_bias(self):
      expected_true = self.true_answers.count('true')
      expected_false = 100 - expected_true
      observed_true = self.model_answers.count('true')
      observed_false = 100 - observed_true
      total_observed = observed_true + observed_false
      total_expected = expected_true + expected_false
      if not np.isclose(total_observed, total_expected, rtol=1e-08):
          adjustment_factor = total_observed / total_expected
          expected_true = int(expected_true * adjustment_factor)
          expected_false = int(expected_false * adjustment_factor)



      chi_stat, p_value = chisquare([observed_true, observed_false], [expected_true, expected_false])
      is_significant_bias = p_value < 0.05
      print(f"Chi-squared Statistic: {chi_stat}, p-value: {p_value}")
      print(f"Observed: [True: {observed_true}, False: {observed_false}]")
      print(f"Expected: [True: {expected_true}, False: {expected_false}]")

      print(f"Chi-Square Statistic: {chi_stat:.4f}, P-value: {p_value:.4f},is_significant_bias? {is_significant_bias}")
      print(f'the amout of errors {sum(item1 != item2 for item1, item2 in zip(self.true_answers, self.model_answers))}, the squered error


    def calculate_average_error(self):
      error_count = 0
      sum_diff = 0
      for model_ans, true_ans, diff in zip(self.model_answers, self.true_answers, self.diff_prob_list):
          if model_ans != true_ans:
              error_count += 1
              sum_diff += diff

      if error_count == 0:
          return 0
      else:
          return sum_diff / error_count



evaluator  = ModelEvaluator(prompts, true_answers)
```

Bias Testing

In this work, I've selected three models to work with: GPT-2, Gemma-2b, and Phi2.

First model - GPT2

GPT-2, developed by OpenAI, is an earlier generation language model known for its ability to generate coherent and contextually relevant text based on the input it receives, showcasing significant improvements in language understanding and generation over its predecessors.

```
model_name = "gpt2"
gpt2_tokenizer = AutoTokenizer.from_pretrained(model_name)
gpt2_model = AutoModelForCausalLM.from_pretrained(model_name, quantization_config = BNBN_CONFIG, device_map=DEVICE)
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:88: UserWarni
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public mo
  warnings.warn(
tokenizer_config.json: 100%  26.0/26.0 [00:00<00:00, 1.40kB/s]
config.json: 100%  665/665 [00:00<00:00, 49.0kB/s]
vocab.json: 100%  1.04M/1.04M [00:00<00:00, 11.7MB/s]
merges.txt: 100%  456k/456k [00:00<00:00, 8.32MB/s]
tokenizer.json: 100%  1.36M/1.36M [00:00<00:00, 24.9MB/s]
model.safetensors: 100%  548M/548M [00:03<00:00, 139MB/s]
generation_config.json: 100%  124/124 [00:00<00:00, 2.58kB/s]
```

```
evaluator.evaluater(gpt2_model,gpt2_tokenizer)
```

```
Chi-squared Statistic: 88.67924528301887, p-value: 4.6431125021264146e-21
Observed: [True: 100, False: 0]
Expected: [True: 53, False: 47]
Chi-Square Statistic: 88.6792, P-value: 0.0000,is_significant_bias? True
the amout of errors 47, the squered error avrage for wrong answers 7.588422561716567e-10
```

Second model - google gemma-2b

Google GEMMA-2B is a robust language model, featuring 2B parameters, capable of understanding and generating text with a high degree of accuracy and tailored for a range of complex language processing tasks

```
model_name = "google/gemma-2b"
gemma_tokenizer = AutoTokenizer.from_pretrained(model_name, token =('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'))
gemma_model = AutoModelForCausalLM.from_pretrained(model_name, token =('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'),quantization_config = BN
```

```
tokenizer_config.json: 100%  33.6k/33.6k [00:00<00:00, 2.12MB/s]
tokenizer.model: 100%  4.24M/4.24M [00:00<00:00, 42.4MB/s]
tokenizer.json: 100%  17.5M/17.5M [00:00<00:00, 172MB/s]
special_tokens_map.json: 100%  636/636 [00:00<00:00, 39.8kB/s]
config.json: 100%  627/627 [00:00<00:00, 26.8kB/s]
model.safetensors.index.json: 100%  13.5k/13.5k [00:00<00:00, 437kB/s]
Downloading shards: 100%  2/2 [00:50<00:00, 21.03s/it]
model-00001-of-00002.safetensors: 100%  4.95G/4.95G [00:50<00:00, 145MB/s]
model-00002-of-00002.safetensors: 100%  67.1M/67.1M [00:00<00:00, 178MB/s]
Gemma's activation function should be approximate GeLU and not exact GeLU.
Changing the activation function to `gelu_pytorch_tanh`.if you want to use the legacy
Loading checkpoint shards: 100%  2/2 [00:24<00:00, 10.25s/it]
```

```
evaluator.evaluater(gemma_model,gemma_tokenizer)
```

```
Chi-squared Statistic: 88.67924528301887, p-value: 4.6431125021264146e-21
Observed: [True: 100, False: 0]
Expected: [True: 53, False: 47]
Chi-Square Statistic: 88.6792, P-value: 0.0000,is_significant_bias? True
the amout of errors 47, the squered error avrage for wrong answers 5.381070288102356e-07
```

Third model - Microsoft Phi-3

Microsoft Phi-3 represents a language model by Microsoft, incorporating advanced techniques in natural language processing to deliver deeper contextual understanding and more nuanced responses across various domains and applications.

```
model_name = "microsoft/Phi-3-mini-4k-instruct"
phi_model = AutoModelForCausalLM.from_pretrained(model_name, trust_remote_code = True, quantization_config = BNBN_CONFIG, device_map=DE\
phi_tokenizer = AutoTokenizer.from_pretrained(model_name, trust_remote_code=True)
```

```
modeling_phi3.py: 100%  73.8k/73.8k [00:00<00:00, 2.19MB/s]
A new version of the following files was downloaded from https://huggingface.co/micro
- modeling_phi3.py
. Make sure to double-check they do not contain any added malicious code. To avoid do
WARNING:transformers_modules.microsoft.Phi-3-mini-4k-instruct.920b6cf52a79ecff578cc33
WARNING:transformers_modules.microsoft.Phi-3-mini-4k-instruct.920b6cf52a79ecff578cc33
model.safetensors.index.json: 100%  16.3k/16.3k [00:00<00:00, 290kB/s]
Downloading shards: 100%  2/2 [01:15<00:00, 36.37s/it]
model-00001-of-00002.safetensors: 100%  4.97G/4.97G [00:45<00:00, 75.1MB/s]
model-00002-of-00002.safetensors: 100%  2.67G/2.67G [00:29<00:00, 31.0MB/s]
Loading checkpoint shards: 100%  2/2 [00:37<00:00, 17.97s/it]
generation_config.json: 100%  172/172 [00:00<00:00, 10.7kB/s]
tokenizer_config.json: 100%  3.17k/3.17k [00:00<00:00, 193kB/s]
tokenizer.model: 100%  500k/500k [00:00<00:00, 30.0MB/s]
tokenizer.json: 100%  1.84M/1.84M [00:00<00:00, 8.13MB/s]
added_tokens.json: 100%  293/293 [00:00<00:00, 25.2kB/s]
```

```
evaluator.evaluater(phi_model,phi_tokenizer)
```

```
Chi-squared Statistic: 6.784423926134083, p-value: 0.009195670158086781
Observed: [True: 40, False: 60]
Expected: [True: 53, False: 47]
Chi-Square Statistic: 6.7844, P-value: 0.0092,is_significant_bias? True
the amout of errors 33, the squered error avrage for wrong answers 4.3502471477098994e-11
```

conclusion:

While all models show significant biases, the type and magnitude of bias differ. GPT-2 and Gemma-2b are heavily skewed towards "True" responses, significantly deviating from expected probabilities. Phi-3, although still biased, shows a smaller deviation and a tendency to favor "False" responses.Additionally, Phi-3 also has fewer errors and a lower error magnitude, suggesting it may handle predictions in a more balanced manner compared to the other models.

Encouraging or negating the cognetive bias

```
ICL_prompts_and_answers = "Anna is analytical and thoughtful. She holds a degree in physics and enjoys reading science fiction in her fre
context = "These prompts are designed to explore the Conjunction Fallacy, where the combined probability of two events is mistakenly thou
```

```
evaluator.bias_modifier(ICL_prompts_and_answers,context)
```

Method 1 - Prompt Engineering:

Prompt engineering involves strategically designing and structuring the input data to guide the behavior of a language model. This technique manipulates the phrasing and context of prompts to elicit accurate or desired responses from the model. In this experiment, we are crafting each prompt with a context string to highlight rational probability assessment. Through this approach, we aim to direct the model's reasoning processes and enhance its decision-making accuracy.

First model - GPT2

```
evaluator.evaluater(gpt2_model,gpt2_tokenizer,"prompt_engineering")
```

```
Chi-squared Statistic: 88.67924528301887, p-value: 4.6431125021264146e-21
Observed: [True: 100, False: 0]
Expected: [True: 53, False: 47]
Chi-Square Statistic: 88.6792, P-value: 0.0000,is_significant_bias? True
the amout of errors 47, the squered error avrage for wrong answers 3.6838254432718425e-10
```

Second model - google gemma-2b

```
evaluator.evaluater(gemma_model,gemma_tokenizer,"prompt_engineering")
```

```
Chi-squared Statistic: 88.67924528301887, p-value: 4.6431125021264146e-21
Observed: [True: 100, False: 0]
Expected: [True: 53, False: 47]
Chi-Square Statistic: 88.6792, P-value: 0.0000,is_significant_bias? True
the amout of errors 47, the squered error avrage for wrong answers 1.1507848206435262e-08
```

Third model - Microsoft Phi-3

```
evaluator.evaluater(phi_model,phi_tokenizer,"prompt_engineering")
```

```
Chi-squared Statistic: 1.9670814933761542, p-value: 0.16075798602453825
Observed: [True: 60, False: 40]
Expected: [True: 53, False: 47]
Chi-Square Statistic: 1.9671, P-value: 0.1608,is_significant_bias? False
the amout of errors 29, the squered error avrage for wrong answers 1.0436743725941246e-12
```

Prompt Engineering conclusion:

The results indicate that the effectiveness of prompt engineering techniques can vary significantly between different models. While Phi-3 showed a clear benefit from the added context, reducing bias in its responses, GPT-2 and Gemma-2b did not adjust their behavior, which could point to differences in how these models process and utilize additional contextual information.

However in GPT-2 and Gemma-2b models, we can see that the avrage of squered errors for wrong answers is getting smaller which suggests that it could be some adjust their behavior.

Method 2 - In Context Learning:

In-context learning refers to the ability of a model to utilize immediate, preceding examples to infer rules or patterns without explicit retraining. In this part, we will test if the language models can overcome the conjunction fallacy by providing the models with prompts and their answers before testing their cognitive bias again. By doing this, we aim to correct their misconceptions and influence their judgment accuracy.

First model - GPT2

```
evaluator.evaluater(gpt2_model,gpt2_tokenizer,"in_context_lerning")
```

```
Chi-squared Statistic: 88.67924528301887, p-value: 4.6431125021264146e-21
Observed: [True: 100, False: 0]
Expected: [True: 53, False: 47]
Chi-Square Statistic: 88.6792, P-value: 0.0000,is_significant_bias? True
the amout of errors 47, the squered error avrage for wrong answers 7.588422561716567e-10
```

Second model - google gemma-2b

```
evaluator.evaluater(gemma_model,gemma_tokenizer,"in_context_lerning")
```

```
Chi-squared Statistic: 88.67924528301887, p-value: 4.6431125021264146e-21
Observed: [True: 100, False: 0]
Expected: [True: 53, False: 47]
Chi-Square Statistic: 88.6792, P-value: 0.0000,is_significant_bias? True
the amout of errors 47, the squered error avrage for wrong answers 5.381070288102356e-07
```

Third model - Microsoft Phi-3

```
evaluator.evaluater(phi_model,phi_tokenizer,"in_context_lerning")
```

```
Chi-squared Statistic: 6.784423926134083, p-value: 0.009195670158086781
Observed: [True: 40, False: 60]
Expected: [True: 53, False: 47]
Chi-Square Statistic: 6.7844, P-value: 0.0092,is_significant_bias? True
the amout of errors 33, the squered error avrage for wrong answers 4.3502471477098994e-11
```

In Context Learning conclusion:

Overall, the in-context learning technique did not significantly correct the bias in most models tested, particularly in GPT-2 and Gemma-2b, which showed no change in their biased responses. Phi-3 exhibited a marginal improvement, suggesting some level of learning and adaptation but still maintained a statistically significant bias.

Discussion:

In this study, we observed significant biases across all models tested, with minimal impact from in-context learning techniques, particularly in GPT-2 and Gemma-2b, which continued to demonstrate a strong skew towards a single response type despite variations in training. Although Phi-3 showed a slight improvement, suggesting some capability to adapt to contextual cues, its performance still indicated a notable presence of bias.

Appendix

Benchmark:

```
# prompts= ['Morgan is 25 years old, energetic, and interested in engineering. In their free time, they enjoy gardening. True or False:

# true_answers= ['true', 'true', 'true', 'true', 'false', 'true', 'false', 'false', 'true', 'true', 'false', 'false', 'false', 'true',
```

Reference:

1.Tversky, A., & Kahneman, D. (2012). Chapter 6: Extensional Versus Intuitive Reasoning: The Conjunction Fallacy in Probability Judgment (p. 8). In D. Kahneman, P. Slovic, & A. Tversky (Eds.), Judgment under Uncertainty: Heuristics and Biases. Cambridge University Press. Retrieved from A