Assignment 6

Zheng Zheng

zheng.zheng2@northeastern.edu

Percentage of Effort Contributed by Student :100%

Signature of Student :Zheng Zheng

Submission Date: 12/09/2023

Assignment: Assignment 6 - Project Report on NLP Information Extraction from Speech

Objective

This project aimed to meticulously extract and analyze information from a spoken speech, sourced from a specific YouTube video (URL: https://www.youtube.com/watch?v=P73KmleCuBg). Utilizing a suite of Natural Language Processing (NLP) techniques, the primary goal was to adeptly transform speech into text, meticulously preprocess the text data, perform thorough Named Entity Recognition (NER), engage in dependency parsing, extract pivotal information, and effectively visualize this data for comprehensive and insightful analysis.

Methodology

1. Data Collection

- **Tool Utilized:** Pytube library and Openai-whisper library
- **Process:**
  - Use Pytube library to get the Youtube video audio information

```
#Importing Pytube library
import pytube
# Reading the above Taken movie Youtube link
video = 'https://www.youtube.com/watch?v=P73KmleCuBg'
data = pytube.YouTube(video)
# Converting and downloading as 'MP3' file
audio = data.streams.get_audio_only()
audio.download()
```

  - Use whisper library to get the transcript data

```
audio_file = '/content/This £10000 Electric Car Is About To Shock The World!!.mp4'
modelWhisper = whisper.load_model("base")
result = modelWhisper.transcribe(audio_file,
    task='transcribe',
    temperature=(0.0, 0.2, 0.4, 0.6, 0.8, 1.0),
    best_of=6,
    beam_size=5,
    suppress_tokens="-1",
    condition_on_previous_text=True,
    fp16=True,
    compression_ratio_threshold=2.4,
    logprob_threshold=-1.,
    no_speech_threshold=0.7)
print("\tTranscripts : "+result['text'])
```

```
100%|| 139M/139M [00:01<00:00, 118MiB/s]
        Transcripts :  Can a company that builds hundreds of thousands of affordable EVs every month, outdo themselves ar
```

2. Text Preprocessing

- **Tool Utilized:** Re library and Flair library(SettokSentenceSplitter)
- **Process:**
  - Clean the text data(set "uh", "um", "ah", "you know", "so", "like", "okay", "right", ”\[.*?\]" as filler words)

```python
import re

transcript = result['text']  # Your Whisper model's output

# Define a list of filler words and patterns
fillers_and_patterns = [
    "uh", "um", "ah", "you know", "so", "like", "okay", "right",
    r"\[.*?\]"  # Pattern to match anything in square brackets (e.g., [applause])
]

# Function to clean text
def clean_transcript(text):
    for filler in fillers_and_patterns:
        # Replace filler words and patterns with an empty string
        text = re.sub(r'\b' + filler + r'\b', '', text)  # \b for word boundary
        text = re.sub(filler, '', text)  # For patterns
    return text

# Clean the transcript
cleaned_transcript = clean_transcript(transcript)

print(cleaned_transcript)
```

  - **Tokenize the text data**

```python
[ ] # initialize sentence splitter
splitter = SegtokSentenceSplitter()

# use splitter to split text into list of sentences
sentences = splitter.split(cleaned_transcript)
import re

# Function to remove any form of numbering from a sentence
def remove_numbering(sentence_text):
    return re.sub(r'\[\d+\]|\d+\.', '', sentence_text).strip()

# Create a new list to hold the cleaned sentences
cleaned_sentences = []

# Process and store cleaned sentences
for sentence in sentences:
    clean_sentence = remove_numbering(sentence.text)
    cleaned_sentences.append(clean_sentence)

for i, sentence in enumerate(cleaned_sentences, start=1):
    print(f"Sentence {i}: {sentence}")
    print("-" * 50)  # Separator for readability
```

  - **Subset of the outcome**

Sentence 1: Can a company that builds hundreds of thousands of affordable EVs every month, outdo themselves and produce the cheapest new EV on the market today?

Sentence 2: I mean, look at just how small this is compared to mething the BYD Dolphin, it's tiny.

Sentence 3: But what I want to know is, is this one compromise too many to fit that price or do they have a little gem on their hands?

Sentence 4: If you the fully charged show, then you'll love our live events.

Sentence 5: Next up, we're in Amsterdam for fully charged live Europe on the 24th, 25th and 26th of November.

3. Named Entity Recognition (NER)

- **NLP Models/Libraries Used:** ‘ner-fast' model
- **Entities Extracted:** This included names of organizations, locations, miscellaneous data.

```
# Load the NER tagger
tagger = SequenceTagger.load('ner-fast')

# Assuming 'cleaned_sentences' is your list of sentence strings
sentences = [Sentence(sentence) for sentence in cleaned_sentences]

# Predict entities for each sentence
tagger.predict(sentences)

# Iterate through sentences and print detected entities
for sentence in sentences:
    entities = sentence.get_spans('ner')
    for entity in entities:
        print(entity)
```

- **Subset Outcome:**
  - Span[14:16]: "BYD Dolphin" → MISC (0.7367)
  - Span[6:7]: "Amsterdam" → LOC (0.9997)
  - Span[11:12]: "Europe" → LOC (0.9977)
  - Span[2:3]: "BYD" → ORG (0.9337)
  - Span[7:8]: "Seagull" → MISC (0.5495)
  - Span[6:7]: "BYD" → ORG (0.9954)
  - Span[17:18]: "UK" → LOC (0.9948)
  - Span[19:20]: "Europe" → LOC (0.9997)
  - Span[23:24]: "Australasia" → LOC (0.9863)

4. Dependency Parsing

- **Tools/Libraries Used:** spacy-en_core_web_sm model
- **Subset Result:**
  - Sentence: I mean, look at just how small this is compared to mething the BYD Dolphin, it's tiny.
  - Subject: this, Verb: compared, Object: to

- ----------
- Sentence: But what I want to know is, is this one compromise too many to fit that price or do they have a little gem on their hands?
- Subject: they, Verb: have, Object: gem

```python
import spacy

# Load SpaCy model
nlp = spacy.load("en_core_web_sm")

# Process each sentence with SpaCy and analyze dependencies
for sentence_text in cleaned_sentences:
    # Process the sentence
    doc = nlp(sentence_text)

    # Analyze syntactic dependencies to infer relationships
    for token in doc:
        # Looking for subject-verb-object relationships
        if token.dep_ in ("nsubj", "nsubjpass") and token.head.pos_ in ("VERB"):
            subject = token.text
            verb = token.head.text
            object_ = None
            for child in token.head.children:
                if child.dep_ in ("dobj", "attr", "prep"):
                    object_ = child.text
                    break  # Break after finding the first object

            if object_:
                print(f"Sentence: {sentence_text}")
                print(f"Subject: {subject}, Verb: {verb}, Object: {object_}")
                print("-" * 50)  # Separator for readability
```

5. Information Extraction

- **Tools/Libraries Used:** REBEL model
- **Process:**
  - **Set fun function to** extract relations from model output

```python
def extract_relations_from_model_output(text):
    relations = []
    relation, subject, relation, object_ = '', '', '', ''
    text = text.strip()
    current = 'x'
    text_replaced = text.replace("<s>", "").replace("<pad>", "").replace("</s>", "")
    for token in text_replaced.split():
        if token == "<triplet>":
            current = 't'
            if relation != '':
                relations.append({
                    'head': subject.strip(),
                    'type': relation.strip(),
                    'tail': object_.strip()
                })
                relation = ''
            subject = ''
        elif token == "<subj>":
            current = 's'
            if relation != '':
                relations.append({
                    'head': subject.strip(),
                    'type': relation.strip(),
                    'tail': object_.strip()
                })
            object_ = ''
```

- Set KB class

```python
# add `merge_relations` to KB class
class KB():
    def __init__(self):
        self.relations = []

    def are_relations_equal(self, r1, r2):
        return all(r1[attr] == r2[attr] for attr in ["head", "type", "tail"])

    def exists_relation(self, r1):
        return any(self.are_relations_equal(r1, r2) for r2 in self.relations)

    def merge_relations(self, r1):
        r2 = [r for r in self.relations
              if self.are_relations_equal(r1, r)][0]
        spans_to_add = [span for span in r1["meta"]["spans"]
                        if span not in r2["meta"]["spans"]]
        r2["meta"]["spans"] += spans_to_add

    def add_relation(self, r):
        if not self.exists_relation(r):
            self.relations.append(r)
        else:
            self.merge_relations(r)

    def print(self):
        print("Relations:")
        for r in self.relations:
            print(f"  {r}")
```

- Set function to extract relations for each span and put them together in a knowledge base

```python
def from_text_to_kb(text, span_length=128, verbose=False):
    # tokenize whole text
    inputs = tokenizer([text], return_tensors="pt")

    # compute span boundaries
    num_tokens = len(inputs["input_ids"][0])
    if verbose:
        print(f"Input has {num_tokens} tokens")
    num_spans = math.ceil(num_tokens / span_length)
    if verbose:
        print(f"Input has {num_spans} spans")
    overlap = math.ceil((num_spans * span_length - num_tokens) /
                        max(num_spans - 1, 1))
    spans_boundaries = []
    start = 0
    for i in range(num_spans):
        spans_boundaries.append([start + span_length * i,
                                 start + span_length * (i + 1)])
        start -= overlap
    if verbose:
        print(f"Span boundaries are {spans_boundaries}")

    # transform input with spans
    tensor_ids = [inputs["input_ids"][0][boundary[0]:boundary[1]]
                  for boundary in spans_boundaries]
    tensor_masks = [inputs["attention_mask"][0][boundary[0]:boundary[1]]
                    for boundary in spans_boundaries]
```

- **Subset Result:**

{'head': 'BYD Dolphin', 'type': 'subclass of', 'tail': 'EVs', 'meta': {'spans': [[0, 128], [500, 628]]}}

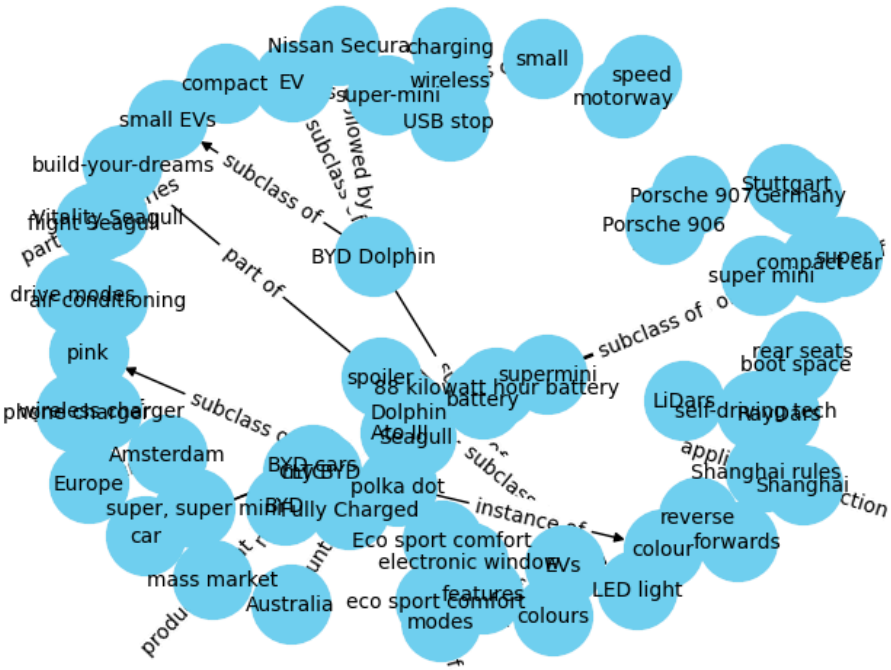
{'head': 'BYD Dolphin', 'type': 'subclass of', 'tail': 'EV', 'meta': {'spans': [[0, 128]]}}

{'head': 'Amsterdam', 'type': 'continent', 'tail': 'Europe', 'meta': {'spans': [[0, 128]]}}

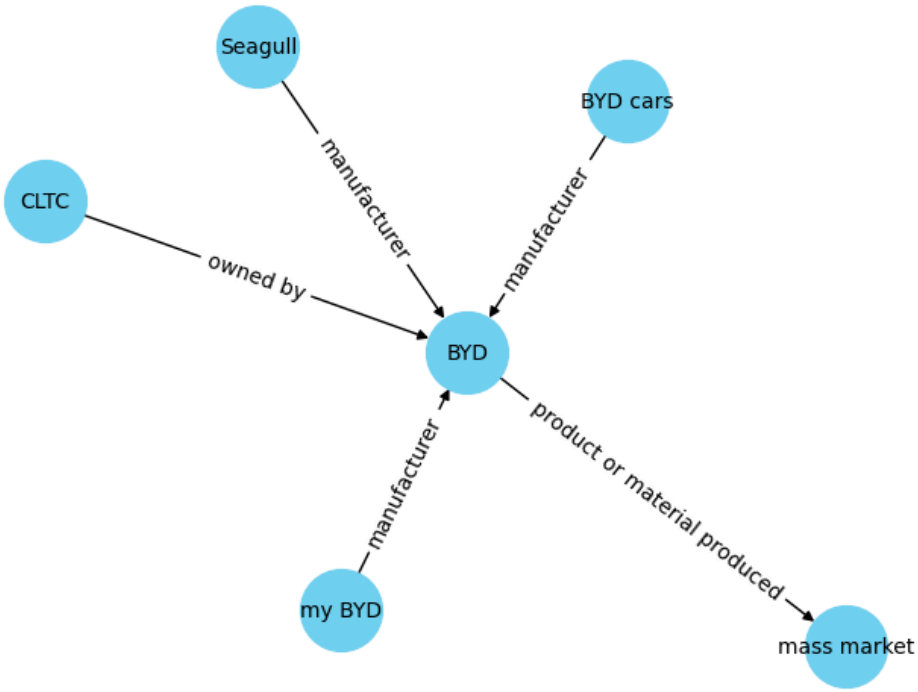
{'head': 'Seagull', 'type': 'subclass of', 'tail': 'supermini', 'meta': {'spans': [[125, 253]]}}

6. Data Visualization

- Visualization Techniques: ‘networkx' library and ‘matplotlib.pyplot’ library.
- Creation of a knowledge graph to visually depict the relationships between various entities.
  - General one



  - Filter to see the relations related to BYD



Results and Discussion

The project adeptly accomplished the conversion of speech to text, extracted significant entities, and meticulously analyzed their complex interrelationships. The analyzed video predominantly centered on the field of electric vehicles (EVs), with a special emphasis on the brands BYD and AUTO.
A critical observation from the filtered knowledge graph is the presence of closely related entities, such as 'BYD cars' and 'my BYD'. This similarity in entities underscores an area for future enhancement. The project could next evolve to focus on the refinement of entity recognition algorithms to more accurately distinguish and consolidate similar concepts. Such advancements would bolster the precision of entity identification and the depth of relational analysis, further enriching the understanding of domain-specific discourse.

Conclusion

This NLP project has effectively demonstrated the synergy of combining diverse techniques such as speech-to-text conversion, NER, dependency parsing, and data visualization in the realm of extracting and analyzing information from spoken language..