CatsMeow2

March 7, 2025

[4]:
```
%%capture
# Cat Meow Translator Notebook - Part 2: AI-Powered Enhancements
# Purrceptron Labs (BETA)

# ------------------
#  0. SET UP AND ACCESS OPEN AI KEY
# ------------------

# -*- coding: utf-8 -*-
"""
MEOW-to-English Translation System v0.2 - Now with AI-Generated Nonsense!
(Still contains mostly AI nonsense, but slightly more amusing nonsense)
"""
!pip install -q librosa matplotlib tensorflow kaggle pandas
!pip install --upgrade openai


import os
import numpy as np
import matplotlib.pyplot as plt
import librosa
import librosa.display
import pandas as pd
from sklearn.model_selection import train_test_split
from tensorflow.keras import layers, models


from google.colab import userdata
# Retrieve the OpenAI API key from userdata (if stored)
openai_api_key = userdata.get('OPENAI_API_KEY')

# If not in userdata, prompt the user to enter it:
if openai_api_key is None:
  openai_api_key = input("Please enter your OpenAI API key: ")

# Set the environment variable
os.environ["OPENAI_API_KEY"] = openai_api_key
```

```
from google.colab import drive
drive.mount('/content/drive')
```

[5]:
```
print("  All systems purr! Let's decode some cat-titude...with AI assistance.")
```

```
 All systems purr! Let's decode some cat-titude…with AI assistance.
```

[6]:
```
# ------------------
# 1. FETCH DATA (Recap from Part 1 - Keeping it brief)
# ------------------

# Upload kaggle.json first!
from google.colab import files
files.upload()  # Upload your kaggle.json

!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
!# CORRECTED DOWNLOAD COMMAND
!kaggle datasets download -d andrewmvd/cat-meow-classification

!unzip -q cat-meow-classification.zip -d cat_meows

# Add this after unzipping
!ls cat_meows  # Check directory structure

print("  Dataset re-acquired. Containing:", len(os.listdir("cat_meows/dataset/
  ↪dataset")), "meows.")
```

```
<IPython.core.display.HTML object>

Saving kaggle.json to kaggle (4).json
Dataset URL: https://www.kaggle.com/datasets/andrewmvd/cat-meow-classification
License(s): Attribution-NonCommercial 4.0 International (CC BY-NC 4.0)
cat-meow-classification.zip: Skipping, found more recently modified local copy
(use --force to force download)
replace cat_meows/dataset/dataset/B_ANI01_MC_FN_SIM01_101.wav? [y]es, [n]o,
[A]ll, [N]one, [r]ename: A
dataset  extras
  Dataset re-acquired. Containing: 440 meows.
```

[7]:
```
# ------------------
# 2. DECODE FILENAMES (Quick Recap)
# ------------------

def parse_cat_filename(filename):
    """Decrypts the feline Da Vinci Code"""
```

```
    parts = filename.split('_')
    return {
        'context': {'B': 'Brushing', 'F': 'Food Demand', 'I':␣
 ↪'Isolation'}[parts[0]],
        'cat_id': parts[1],
        'breed': 'Maine Coon' if parts[2] == 'MC' else 'European Shorthair',
        'sex': parts[3],
        'owner_id': parts[4],
        'session': parts[5][1:],
        'counter': parts[6].split('.')[0]
    }
```

[8]:
```
# ------------------
# 3. AUDIO PREPROCESSING (Quick Recap)
# ------------------

def load_and_process(file_path):
    """Converts meows to machine-digestible format"""
    try:
        audio, sr = librosa.load(file_path, sr=16000)  # Downsample to 16kHz
        mfccs = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=13)
        return mfccs.T
    except Exception as e:
        print(f" Failed to process {file_path}: {str(e)}")
        return None
```

[9]:
```
# ------------------
# 4. PREPARE DATA (Pad Sequences - Just Loading, no new preprocessing)
# ------------------

X = []
y = []
context_map = {'B': 0, 'F': 1, 'I': 2}
reverse_map = {v:k for k,v in context_map.items()}

for file in os.listdir("cat_meows/dataset/dataset"):
    if file.endswith(".wav"):
        file_path = os.path.join("cat_meows/dataset/dataset", file)
        mfccs = load_and_process(file_path)
        if mfccs is not None:  # Check if MFCCs are valid
            X.append(mfccs)
            y.append(context_map[file[0]])

# Pad sequences to same length
max_len = max([x.shape[0] for x in X])
X_padded = np.array([np.pad(x, ((0,max_len - x.shape[0]),(0,0))) for x in X])
y = np.array(y)
```

```
print(f"  Data shape: {X_padded.shape} | Meow contexts: {np.unique(y,␣
 ↪return_counts=True)}")
```

```
 Data shape: (440, 126, 13) | Meow contexts: (array([0, 1, 2]), array([127,
92, 221]))
```

[10]:
```
# ------------------
# 5. LOAD Pre-Trained Model (Quickly just load the data and model)
# ------------------
from tensorflow.keras.models import load_model

model = load_model('/content/drive/MyDrive/cat_meow_model.keras')
```

[11]:
```
# ------------------
# 6. Set up AI Translations Dictionary and Cat Personas
# ------------------

class CatTranslator:
    def __init__(self, model, context_map):
        self.model = model
        self.reverse_map = reverse_map
        self.translations = {
            0: ["Human. You disturb my fur.", "The brush displeases me.", "This␣
 ↪grooming is acceptable."],
            1: ["FOOD NOW.", "I smell tuna. Provide it.", "The bowl is EMPTY,␣
 ↪peasant."],
            2: ["WHERE IS EVERYONE?!", "This place smells wrong.", "I demand␣
 ↪cuddles immediately!"]
        }

    def translate(self, file_path):
        features = load_and_process(file_path)
        padded = np.pad(features, ((0,max_len - features.shape[0]),(0,0)))
        pred = model.predict(padded[np.newaxis, ...], verbose=0) # reduce␣
 ↪logging
        context = self.reverse_map[np.argmax(pred)]
        return np.random.choice(self.translations[np.argmax(pred)]) # only␣
 ↪translations from the CatTranslator are here


# Define the Cat Personas (Crucial for OpenAI)
cat_personas = {
    "sarcastic_maine_coon": "A sarcastic, demanding Maine Coon with a dry wit,␣
 ↪who uses understatement and passive-aggressive remarks.  They secretly love␣
 ↪their humans, but would NEVER admit it.",
```

```
    "melodramatic_siamese": "A melodramatic Siamese cat, prone to exaggeration␣
 ↪and expressing everything as a matter of life and death. Thinks a slight␣
 ↪inconvenience is a personal tragedy and expresses emotions in high drama.",
    "grumpy_persian": "A grumpy, introverted Persian who mostly wants to be␣
 ↪left alone and is easily annoyed by EVERYTHING. Uses a lot of sarcasm and␣
 ↪insults, but in a very deadpan way.",
    "poetic_bengal": "A poetic Bengal who expresses everything in vivid imagery␣
 ↪and metaphors, often comparing household objects to grand, natural␣
 ↪landscapes or mythical creatures. Very self aware and arrogant."
}

# Function to generate more translations with OpenAI
def generate_cat_translations(behavior, translations, cat_persona,␣
 ↪num_translations=3):
    """Generates more cat translations using OpenAI, given a behavior,
    initial translations, and a cat persona."""
    from openai import OpenAI
    client = OpenAI()

    prompt = f"""I want you to act as a professional cat translator,
    speaking for a cat with the following persona: {cat_persona}.
    The cat is exhibiting the behavior: {behavior}.

    I want you to act as a professional cat translator with the following␣
 ↪sample. Translate the prompt into a persona cat as follows.
    'I am exhibiting the behavior: Hunger' and some example translations are:
    - I want food, please!' responds, 'The bowl is empty. This displeases me'.
    -I need tuna and I need it now!" responds with, 'I smell tuna. Provide it.'.
    -My needs aren't being met!" responds with, 'The bowl is as barren as the␣
 ↪Sahara, RECTIFY THIS ATROCITY!'

    I have a few examples of what the cat might say: {translations}.
    Give me {num_translations} MORE unique and creative translations, fitting␣
 ↪the persona. Provide the translations as a JSON list.  Return only the json.␣
 ↪ Do not wrap the response in ```json or backticks."""

    response = client.chat.completions.create(
        model="gpt-4o",
        messages=[{"role": "user", "content": prompt}],
    )
    # Clean up the string
    new_translations = response.choices[0].message.content

    # Load the string as a JSON list
    import json
    # Added error handling for json.loads()
```

```
    try:
        new_translations = json.loads(new_translations)
    except json.JSONDecodeError as e:
        print(f"Error decoding JSON: {e}")
        print(f"Raw response: {new_translations}")
        return []  # Return an empty list if decoding fails

    cleaned_translations = [t.strip() for t in new_translations if t.strip()]
    return cleaned_translations

# AI translation can be applied to the translation lists as below
def augment_translations(translations, cat_persona):

  for key in translations.keys():
    new_translations = generate_cat_translations(reverse_map[key],␣
 ↪translations[key], cat_persona, num_translations=5) #get 5 new ones
    for new_translation in new_translations:
        translations[key].append(new_translation) #add to old
  return translations

# Randomly Select a Persona
persona_key = np.random.choice(list(cat_personas.keys()))
selected_persona = cat_personas[persona_key]
print(f"  Selected cat persona: {persona_key}")

# Generate a larger dictionary, including more starter material
ai_translations = {
    0: [],
    1: [],
    2: []
}
ai_translations = augment_translations(ai_translations, selected_persona) #␣
 ↪Apply translations from the AI now

# Instantiate CatTranslator with the generated translations
class CatTranslatorExtended(CatTranslator): # reuses the trained model, but␣
 ↪adds AI translations
    def __init__(self, model, context_map, ai_translations):
        super().__init__(model, context_map)
        self.ai_translations = ai_translations # Load ai translations

    def translate(self, file_path):
        features = load_and_process(file_path)
        if features is None:
            return "Error: Could not process audio."  # Handle the case where␣
 ↪audio processing fails
        padded = np.pad(features, ((0, max_len - features.shape[0]), (0, 0)))
```

```
        pred = self.model.predict(padded[np.newaxis, ...], verbose=0)  # Pass␣
 ↪verbose=0 to prevent training logs during prediction
        context = self.reverse_map[np.argmax(pred)]
        # Here the choice of the translation will be from a much longer␣
 ↪translation, due to augment_translations()
        return np.random.choice(self.ai_translations[np.argmax(pred)]) # this␣
 ↪line is the difference!

translator = CatTranslatorExtended(model, context_map, ai_translations) #␣
 ↪create the translation with the updated ai_translations


print(translator.translations)
```

```
 Selected cat persona: sarcastic_maine_coon
{0: ['Human. You disturb my fur.', 'The brush displeases me.', 'This grooming is
acceptable.'], 1: ['FOOD NOW.', 'I smell tuna. Provide it.', 'The bowl is EMPTY,
peasant.'], 2: ['WHERE IS EVERYONE?!', 'This place smells wrong.', 'I demand
cuddles immediately!']}
```

[12]:
```
# Run a Test with new and Improved translations
# test_meow = "cat_meows/dataset/dataset/" + np.random.choice(os.
 ↪listdir("cat_meows/dataset/dataset/"))
# from IPython.display import Audio, display
# display(Audio(test_meow))
# print(f" Playing: {test_meow}")
# print(f" AI Translation: {translator.translate(test_meow)}")

test_meow = "cat_meows/dataset/dataset/" + np.random.choice(os.
 ↪listdir("cat_meows/dataset/dataset/"))
from IPython.display import Audio, display
display(Audio(test_meow))
print(f" Playing: {test_meow}")
print(f" AI Translation: {translator.translate(test_meow)}")
```

```
<IPython.lib.display.Audio object>
```

```
 Playing: cat_meows/dataset/dataset/B_NUL01_MC_MI_SIM01_301.wav
 AI Translation: I could fade into a shadow from this enforced fasting.
```

[ ]:
```
!jupyter nbconvert --to html /content/CatsMeow.ipynb
```

[ ]:
```
!pip install nbconvert
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc
!jupyter nbconvert --to pdf /content/CatsMeow.ipynb
```

[14]:
```
```