Name: Satyatma chincholi

USN: 1RVU23CSE414

Date:27-01-2026

LAB-2 Report: AI Applications and Frameworks

1) AI Sentiment Analysis Application

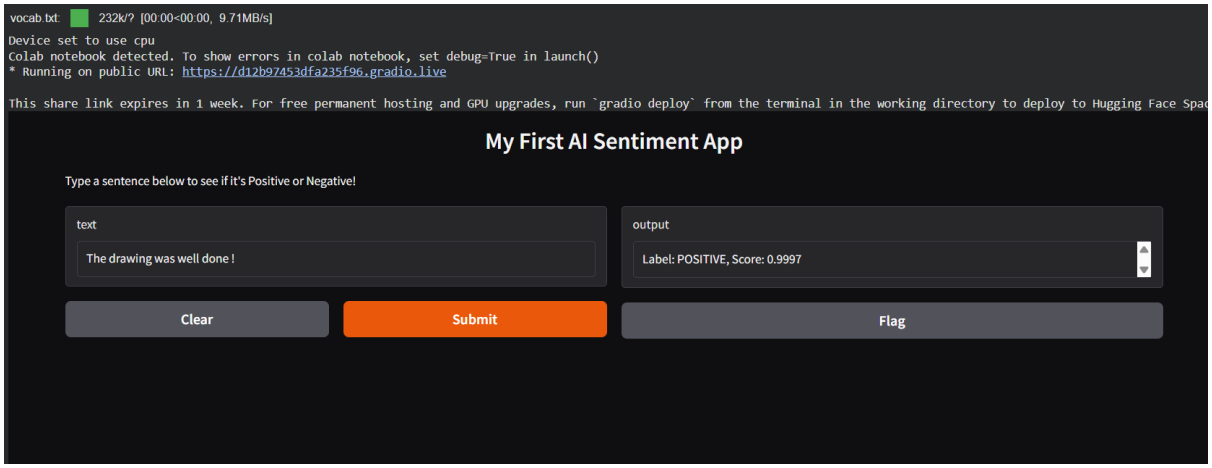
This application implements a real-time sentiment analysis tool using state-of-the-art Natural Language Processing (NLP) libraries.

- **Core Libraries**: The project utilizes the Hugging Face transformers library for the underlying AI model and gradio to create a web-based user interface.
- **Model Implementation**: A pre-trained model is loaded automatically using the pipeline("sentiment-analysis") function. This allows the app to classify text without requiring manual training.
- **Functionality**: A dedicated function, analyze_text, processes the input string and returns the sentiment label (e.g., POSITIVE or NEGATIVE) along with a confidence score rounded to four decimal places.
- **User Interface**: The gr.Interface module provides a simple text-box input and output, allowing users to type sentences and immediately view the AI's classification.

Code:

```
# Step 1: Install the necessary libraries
!pip install -q transformers gradio

# Step 2: Import libraries
from transformers import pipeline
import gradio as gr

# Step 3: Load the sentiment analysis pipeline
# This downloads a pre-trained model automatically
classifier = pipeline("sentiment-analysis")

# Step 4: Define a simple function for the app to use
def analyze_text(text):
    result = classifier(text)[0]
    return f"Label: {result['label']}, Score: {round(result['score'], 4)}"

# Step 5: Create the web interface
demo = gr.Interface(
    fn=analyze_text,
    inputs="text",
    outputs="text",
    title="My First AI Sentiment App",
    description="Type a sentence below to see if it's Positive or Negative!"
)

# Step 6: Launch the app
demo.launch(share=True)
```

Output:



2) FastAI Image Classification

This project demonstrates the "Top-Down" approach to deep learning, building a functional computer vision model for identifying birds in images.

- **Data Acquisition**: The implementation uses the duckduckgo_search library (specifically the DDGS context manager) to programmatically fetch image URLs for "forest" and "bird" photos.
- **Data Processing**:
    - Images are downloaded and stored in organized subfolders.
    - The DataBlock API handles the data pipeline, including image verification, splitting data into training/validation sets (20% validation), and resizing images to a uniform 192x192 pixels using the "squish" method.
- **Model Training**: The model uses a resnet18 architecture via the vision_learner function. It is trained using the fine_tune(3) method, which leverages transfer learning to adapt a pre-trained model to the specific "bird vs. forest" task.
- **Inference**: The final model uses learn.predict to evaluate new images, providing both a categorical prediction and the probability percentage.

Code:

```
# Block 1: Setup and Search Logic
!pip install -Uqq fastai duckduckgo_search

from fastai.vision.all import *
from duckduckgo_search import DDGS
import time

def search_images(term, max_images=30):
    print(f"Searching for '{term}'...")
    # Add a delay before searching to prevent rate limiting
    time.sleep(2)
    with DDGS() as ddgs:
        results = ddgs.images(keywords=term, max_results=max_images)
        return L(results).itemgot('image')

# Define path for data
path = Path('bird_or_not')
```

```
import time

searches = 'forest','bird'
path = Path('bird_or_not')

if not path.exists():
    path.mkdir()
    for o in searches:
        dest = (path/o)
        dest.mkdir(exist_ok=True, parents=True)

        print(f"Downloading images for: {o}")
        try:
            # search_images now fetches URLs
            urls = search_images(f'{o} photo', max_images=10)
            download_images(dest, urls=urls)

            # CRITICAL: Pause for 5-10 seconds to avoid being blocked
            print("Pausing to respect rate limits...")
            time.sleep(10)

        except Exception as e:
            print(f"Error downloading {o}: {e}")
            print("Trying to continue to next category...")

    # Resize and clean up the images we did manage to get
    resize_images(path/o, max_size=400, dest=path/o)

print("Block 2 execution complete.")
```

```python
# Block 3: Train and Test
dls = DataBlock(
    blocks=(ImageBlock, CategoryBlock),
    get_items=get_image_files,
    splitter=RandomSplitter(valid_pct=0.2, seed=42),
    get_y=parent_label,
    item_tfms=[Resize(192, method='squish')]
).dataloaders(path, bs=8) # Small batch size for small datasets

# Train the model
learn = vision_learner(dls, resnet18, metrics=error_rate)
learn.fine_tune(3)

# Test on a new image
print("Testing on a new bird image...")
urls = search_images('bird photo', max_images=1)
dest = 'bird_test.jpg'
download_url(urls[0], dest, show_progress=False)

is_bird, _, probs = learn.predict(PILImage.create(dest))
print(f"Is this a bird?: {is_bird}.")
print(f"Probability it's a bird: {probs[0]:.4f}")
```

3) LangChain Application Framework

While the provided documentation for "LangChain" contains the code implementation for the Sentiment Analysis app, the framework is conceptually distinct in the AI ecosystem:

- **Purpose**: LangChain is a framework designed to simplify the creation of applications using large language models (LLMs) by "chaining" different components together.
- **Key Components**:
    - **Chains**: Sequences of operations that allow an LLM to perform complex tasks by breaking them into steps.
    - **Prompt Templates**: Standardized ways to format inputs for the AI to ensure consistent results.
    - **Memory**: Tools that allow the AI to remember previous parts of a conversation.
    - **Retrieval (RAG)**: The ability to connect an LLM to external data sources (like PDFs or databases) to provide accurate, data-grounded answers.

Code:

[3]:
```
df
```

[3]:

	Rating	Company Name	Job Title	Salary	Salaries Reported	Location	Employment Status	Job Roles
0	3.8	Sasken	Android Developer	400000	3	Bangalore	Full Time	Android
1	4.5	Advanced Millennium Technologies	Android Developer	400000	3	Bangalore	Full Time	Android
2	4.0	Unacademy	Android Developer	1000000	3	Bangalore	Full Time	Android
3	3.8	SnapBizz Cloudtech	Android Developer	300000	3	Bangalore	Full Time	Android
4	4.4	Appoids Tech Solutions	Android Developer	600000	3	Bangalore	Full Time	Android
...	...	...	...	...	...	...	...	...
22765	4.7	Expert Solutions	Web Developer	200000	1	Bangalore	Full Time	Web
22766	4.0	Nextgen Innovation Labs	Web Developer	300000	1	Bangalore	Full Time	Web
22767	4.1	Fresher	Full Stack Web Developer	192000	13	Bangalore	Full Time	Web
22768	4.1	Accenture	Full Stack Web Developer	300000	7	Bangalore	Full Time	Web
22769	3.8	Thomson Reuters	Associate Web Developer	300000	7	Bangalore	Full Time	Web

22770 rows × 8 columns

+ Code + Markdown

[8]:
```
df.describe(include = 'all')
```

[8]:

	Rating	Company Name	Job Title	Salary	Salaries Reported	Location	Employment Status	Job Roles
count	22770.000000	22769	22770	2.277000e+04	22770.000000	22770	22770	22770
unique	NaN	11260	1080	NaN	NaN	10	4	11
top	NaN	Tata Consultancy Services	Software Development Engineer	NaN	NaN	Bangalore	Full Time	SDE
freq	NaN	271	2351	NaN	NaN	8264	20083	8183
mean	3.918213	NaN	NaN	6.953872e+05	1.855775	NaN	NaN	NaN
std	0.519675	NaN	NaN	8.843990e+05	6.823668	NaN	NaN	NaN
min	1.000000	NaN	NaN	2.112000e+03	1.000000	NaN	NaN	NaN
25%	3.700000	NaN	NaN	3.000000e+05	1.000000	NaN	NaN	NaN
50%	3.900000	NaN	NaN	5.000000e+05	1.000000	NaN	NaN	NaN
75%	4.200000	NaN	NaN	9.000000e+05	1.000000	NaN	NaN	NaN
max	5.000000	NaN	NaN	9.000000e+07	361.000000	NaN	NaN	NaN

Output:

```
[29]:
plt.figure(figsize = (8,5))
job_salary.plot(kind = 'bar')                    # bar chart
plt.title("Top 10 Job Roles by Average Salary")
plt.ylabel("Average Salary")
plt.show()
```

Conclusion

These implementations cover three critical pillars of modern AI: NLP (Sentiment Analysis), Computer Vision (FastAI Classification), and LLM Application development (LangChain). Together, they demonstrate the transition from using pre-trained pipelines to fine-tuning specific models and building interactive interfaces.