C.Ramaswaroop **24010107003**

Semantic Profiling

Based on the uploaded presentation, with Python code examples

Semantic profiling is the process of analyzing data, text, code, or datasets to understand what they really are and what they represent. In one line: **Understanding and extracting meaning from data.**

1. Basic Example

Sentence: **"Apple released a new iPhone."**
Semantic profile: Apple → company/organization; released → action; new iPhone → product; overall meaning → Apple launched a new product.

Python Code — Basic Semantic Profiling

```
text = "Apple released a new iPhone."

profile = {
    "Apple": "company/organization",
    "released": "action",
    "new iPhone": "product",
    "overall meaning": "Apple launched a new product"
}

for item, meaning in profile.items():
    print(f"{item} -> {meaning}")
```

2. Semantic Profiling in NLP

The presentation identifies these NLP components: meaning of words, relationships between words, entities such as people/companies/places/products, context, intent behind a sentence, and sentiment (positive, negative, or neutral).

Python Code — Simple NLP Semantic Profile

```
text = "I loved the movie. The acting was excellent!"

semantic_profile = {
    "Topic": "Movie",
    "Sentiment": "Positive",
    "Opinion": "Loved",
    "Attribute": "Acting",
    "Evaluation": "Excellent"
}

for key, value in semantic_profile.items():
    print(f"{key}: {value}")
```

3. Sentiment Profiling

For the sentence **"I loved the movie. The acting was excellent!"**, the presentation gives the topic as Movie, sentiment as Positive, opinion as Loved, attribute as Acting, and evaluation as Excellent.

Python Code — Simple Sentiment Classification

```
text = "I loved the movie. The acting was excellent!"

positive_words = ["loved", "excellent", "good", "great", "beautiful"]
negative_words = ["poor", "bad", "worst", "terrible"]
```

```
words = text.lower().replace(".", "").replace(",", "").split()

positive_count = sum(word in positive_words for word in words)
negative_count = sum(word in negative_words for word in words)

if positive_count > negative_count:
    sentiment = "Positive"
elif negative_count > positive_count:
    sentiment = "Negative"
else:
    sentiment = "Neutral"

print("Sentiment:", sentiment)
```

4. Word Frequency Graph

The presentation includes semantic profiling using word frequency by graph. The following Python code creates a simple frequency graph from a sentence.

```
import matplotlib.pyplot as plt
from collections import Counter

text = "I loved the movie. The acting was excellent!"
words = text.lower().replace(".", "").replace("!", "").split()

frequency = Counter(words)

plt.bar(frequency.keys(), frequency.values())
plt.xlabel("Words")
plt.ylabel("Frequency")
plt.title("Semantic Profiling - Word Frequency")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

5. Tree Relationship

The presentation gives the example: **"I love this phone. The camera is excellent, but the battery is poor."** Its tree representation is shown as Phone Review with Camera → Excellent, Display → Beautiful, Battery → Poor, and Performance → Very Good.

Python Code — Tree Representation

```
class Node:
    def __init__(self, name):
        self.name = name
        self.children = []

    def add_child(self, child):
        self.children.append(child)

    def display(self, level=0):
        print("  " * level + self.name)
        for child in self.children:
            child.display(level + 1)

root = Node("Phone Review")

camera = Node("Camera")
camera.add_child(Node("Excellent"))

display = Node("Display")
display.add_child(Node("Beautiful"))

battery = Node("Battery")
battery.add_child(Node("Poor"))

performance = Node("Performance")
performance.add_child(Node("Very Good"))
```

```
root.add_child(camera)
root.add_child(display)
root.add_child(battery)
root.add_child(performance)

root.display()
```