# WEEK 3 AI FOR SOFTWARE ENGINEERING ASSIGNMENT

## PART 1

### QUESTION 1

**Q1. The primary differences between TensorFlow and PyTorch and when to choose one over the other.**

### 1. Programming Style

- **PyTorch**: Imperative (eager execution) – you write code and execute it line by line. Feels more "Pythonic" and intuitive.
- **TensorFlow**: Originally static graph-based (define-and-run), though **eager execution** is now supported (like PyTorch).

**Choose PyTorch** if you prefer easier debugging and a more intuitive coding experience.

---

### 2. Model Deployment

- **TensorFlow**: Strong deployment ecosystem – supports **TensorFlow Lite**, **TensorFlow Serving**, **TensorFlow.js**, and **TFHub**.
- **PyTorch**: Uses **TorchServe** and **ONNX**, but its ecosystem for deployment is relatively newer.

**Choose TensorFlow** if you want seamless **cross-platform model deployment** (mobile, web, embedded).

---

### 3. Community and Ecosystem

- **TensorFlow**: Backed by **Google**; has a mature ecosystem and extensive tooling (e.g., **TensorBoard**, **TFX**).
- **PyTorch**: Backed by **Meta (Facebook)**; growing rapidly in research and production usage.

**Choose PyTorch** if you're in **academic/research settings**, TensorFlow for **production-grade enterprise solutions**.

---

### 4. Model Serving and Production Pipelines

- **TensorFlow**: More robust tools for end-to-end production pipelines (TFX, Data Validation, etc.).
- **PyTorch**: Easier prototyping, but production support catching up.

**Choose TensorFlow** for **enterprise-scale MLOps**; PyTorch for fast prototyping and iterative development.

---

**5. Performance**

- Both have high performance and GPU acceleration via CUDA.
- TensorFlow has better **multi-GPU and TPU support** out of the box.

**Choose TensorFlow** if you're leveraging **TPUs or distributed training**.

---

**Summary Decision Guide:**

| Use Case | Recommendation |
|---|---|
| Rapid prototyping & research | **PyTorch** |
| Large-scale production deployment | **TensorFlow** |
| Beginners learning deep learning | **PyTorch** |
| Mobile or embedded AI apps | **TensorFlow** |
| Cutting-edge AI research papers | **PyTorch** |

**Q2. Two use cases for Jupyter Notebooks in AI development.**

Introduction

Jupyter Notebooks offer an interactive environment that seamlessly integrates code, data visualizations, and narrative text, making them invaluable for AI development. Below are two detailed use cases:

---

**1. Exploratory Data Analysis (EDA) and Preprocessing**

**Overview:**
When building AI models, understanding and preparing your data is crucial. Jupyter Notebooks facilitate exploratory data analysis by allowing you to:

- **Interactive Data Exploration:**
  Load and view datasets using libraries like Pandas and NumPy, and then generate

dynamic visualizations with tools such as Matplotlib, Seaborn, or Plotly. This interactivity helps in identifying patterns, outliers, or trends within the data.

- **Step-by-Step Data Cleaning:**
  Experiment with different data cleaning techniques interactively (e.g., handling missing data, normalization, feature scaling, and feature selection). You can iteratively test and refine your preprocessing pipeline within the same document, which is invaluable for tailoring your dataset to the needs of your AI model.
- **Documentation and Sharing:**
  Combine code with markdown cells to document insights, hypotheses, and methodology. This makes it easier to share your findings with team members or stakeholders and maintain reproducibility of your data preparation process.

**When It's Useful:**

- During the early stages of model development, where understanding the data is key.
- For collaborative projects, where clear communication of the data cleaning steps and insights is needed.

---

### 2. Prototyping, Experimenting, and Model Iteration

**Overview:**
Jupyter Notebooks serve as an ideal platform for rapid prototyping and iterative experimentation with AI models. They allow you to:

- **Model Development and Experimentation:**
  Write and execute code in small, manageable blocks, enabling you to quickly develop, test, and refine machine learning models. Libraries such as TensorFlow, PyTorch, and Scikit-learn can be integrated directly into your notebook cells.
- **Visualization of Model Outputs:**
  Immediately visualize the predictions, loss curves, and performance metrics as your model trains or when comparing different models. This immediate feedback loop is crucial for adjusting hyperparameters, diagnosing issues like overfitting, and enhancing overall model performance.
- **Integration of Narrative and Code:**
  Annotate experiments with explanations, record observations, and highlight performance improvements using markdown cells. This helps create a clear narrative of the model development process and serves as a reference for future iterations.
- **Reproducibility and Collaboration:**
  Jupyter Notebooks can be version-controlled and shared among colleagues. This collaborative approach ensures that each experiment is documented and can be reproduced or built upon by other team members, fostering an efficient and creative development cycle.

**When It's Useful:**

- In research or academic settings where iterative experimentation and documentation are vital.
- In industry projects that benefit from rapid prototyping and collaborative debugging sessions.

---

### Conclusion

These two use cases highlight the flexibility and power of Jupyter Notebooks in AI development, supporting both the foundational tasks of data exploration and the dynamic requirements of model prototyping and iteration.

## Q3. How spaCy enhance NLP tasks compared to basic Python string operations.

### Introduction

spaCy significantly enhances NLP (Natural Language Processing) tasks compared to basic Python string operations by providing **linguistically intelligent**, **efficient**, and **scalable** tools for analyzing and processing human language. Here's how:

---

### 1. Tokenization and Text Structure Understanding

- **Basic Python**: Uses. split () or regex, which splits text naively by whitespace or punctuation.
- **spaCy**: Performs *intelligent tokenization* — understanding contractions (e.g., "can't" → "ca" + "n't"), punctuation, and special cases using built-in language models.

**Benefit**: More accurate parsing of real-world language structures.

---

### 2. Part-of-Speech (POS) Tagging

- **Basic Python**: Has no built-in POS tagging capability.
- **spaCy**: Automatically labels each word with its grammatical role (e.g., noun, verb, adjective).

**Benefit**: Enables syntax-aware processing — useful in tasks like grammar checking, information extraction, or question answering.

---

### 3. Named Entity Recognition (NER)

- **Basic Python**: Requires manual rule-based approaches or keyword matching.
- **spaCy**: Identifies and classifies named entities (e.g., people, organizations, dates) out of the box.

**Benefit**: Extracts meaningful data (e.g., “Barack Obama” → PERSON) with high accuracy and minimal effort.

---

### 4. Dependency Parsing

- **Basic Python**: Lacks understanding of grammatical relationships.
- **spaCy**: Constructs dependency trees to show how words relate (e.g., subject → verb → object).

**Benefit**: Crucial for complex NLP tasks like semantic role labeling, question generation, or text summarization.

---

### 5. Lemmatization

- **Basic Python**: Requires custom dictionaries or stemming libraries.
- **spaCy**: Reduces words to their base forms (e.g., "running" → "run") using linguistic context.

**Benefit**: Improves search, matching, and classification performance by normalizing text.

---

### 6. Pretrained Models and Language Support

- **Basic Python**: Manual implementation for every task.
- **spaCy**: Comes with pretrained models for multiple languages and tasks, optimized for performance and accuracy.

**Benefit**: Quick access to powerful NLP pipelines without building from scratch.

---

**Summary Comparison**

| Task | Basic Python | spaCy |
|---|---|---|
| Tokenization | str.split() / regex | Context-aware and accurate |
| POS Tagging | Not available | Built-in with pretrained models |
| Named Entity Recognition | Manual | Automatic, high-accuracy |
| Dependency Parsing | Not available | Built-in syntactic parsing |
| Lemmatization | Manual or NLTK-based | Fast and linguistically correct |
| Performance | Slower with large text | Highly optimized for speed & scale |

---

**Conclusion:**
spaCy elevates text processing from raw string manipulation to true language understanding, enabling more accurate and powerful NLP applications like chatbots, text classifiers, summarizers, and knowledge extractors.

**QUESTION 2**

## 1. Target Applications

| Aspect | Scikit-learn | TensorFlow |
|---|---|---|
| **Focus Area** | Classical Machine Learning | Deep Learning / Neural Networks |
| **Common Use Cases** | Regression, classification, clustering, dimensionality reduction, model evaluation | Image processing, NLP, speech recognition, large-scale neural networks |
| **Model Types** | Decision trees, SVMs, random forests, k-NN, logistic regression | CNNs, RNNs, Transformers, GANs, custom DNNs |

- **Use Scikit-learn** for traditional ML tasks on structured/tabular data.
- **Use TensorFlow** for complex tasks like computer vision, natural language processing, and large-scale AI.

## 2. Ease of Use for Beginners

| Aspect | Scikit-learn | TensorFlow |
| --- | --- | --- |
| **API Design** | Simple, consistent, and intuitive | More complex, especially in low-level APIs |
| **Learning Curve** | Gentle – great for beginners | Steeper – especially before TensorFlow 2.0 |
| **Workflow Simplicity** | Fit/predict pipeline is beginner-friendly | Keras API improves ease, but deep learning remains more complex |

- **Scikit-learn is easier** for beginners with basic programming or data science backgrounds.
- **TensorFlow (via Keras)** can also be accessible but is better for learners with some understanding of neural networks.

---

## 3. Community Support

| Aspect | Scikit-learn | TensorFlow |
| --- | --- | --- |
| **Community Size** | Large and well-established in academia | Very large, especially in industry and research |
| **Documentation** | Excellent and beginner-friendly | Extensive, especially with TensorFlow 2.x and Keras |
| **Ecosystem** | Integrates well with Pandas, NumPy, etc. | Strong ecosystem: TensorBoard, TF Lite, TF Hub, etc. |
| **Online Resources** | Abundant tutorials and examples | Very active forums, GitHub issues, and courses |

- **Both have strong communities**, but TensorFlow has broader industrial backing and tools for production deployment.
- Scikit-learn is **more common in traditional ML academic settings**, while TensorFlow is a **powerhouse for AI in production**.

---

## 🔍 Summary Table:

| Feature | Scikit-learn | TensorFlow |
| --- | --- | --- |
| **Target Applications** | Classical ML | Deep Learning & AI |
| **Best for Beginners** | Very beginner-friendly | Moderate (easier with Keras) |
| **Community Support** | Strong in ML & data science | Massive in AI and industry |

### Recommendation:

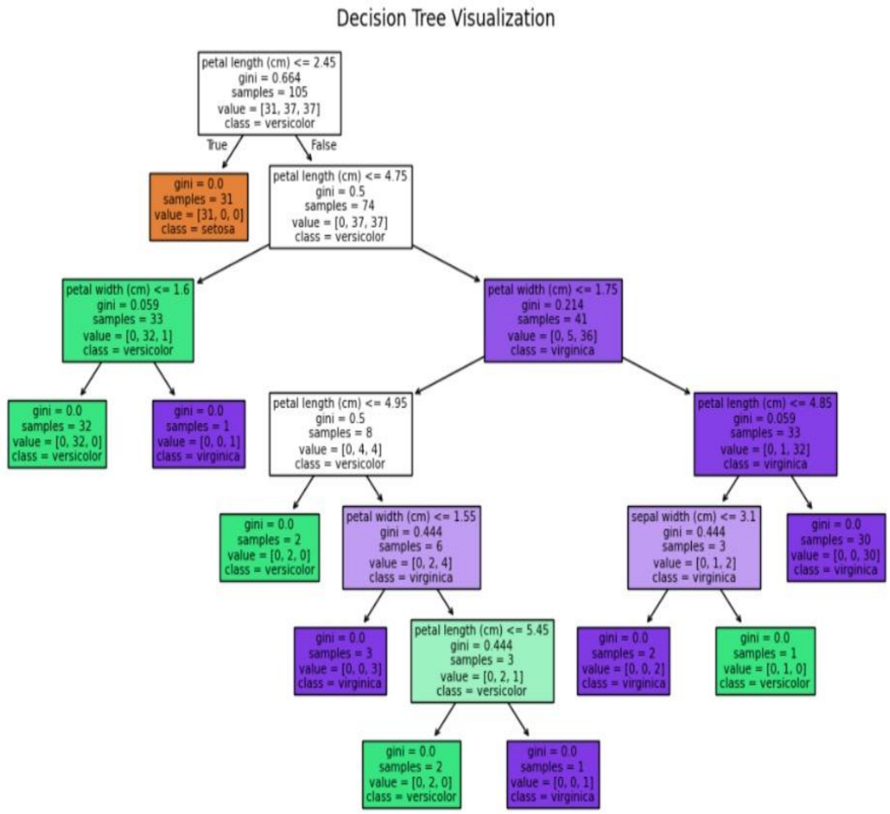
- Start with **Scikit-learn** if you're new to ML and working with structured data.
- Move to **TensorFlow** when you're ready to build and train deep learning models for complex tasks like image recognition or NLP.

## SCREENSHORTS

```
🔍 Named Entity Recognition (NER):

Review 1: I love my new Samsung Galaxy S21, the camera is amazing!
 - Samsung Galaxy S21 (ORG)
------------------------------------------------------------
Review 2: The Apple AirPods Pro are so comfortable and the sound is top-notch.
 - No recognizable product/brand entity found.
------------------------------------------------------------
Review 3: I was disappointed with the Sony headphones, poor noise cancellation.
 - Sony (ORG)
------------------------------------------------------------
Review 4: This HP laptop is very fast and reliable for work and school.
 - HP (ORG)
------------------------------------------------------------
Review 5: Avoid the fake Ray-Ban sunglasses - cheap material and broke in 2 days!
 - No recognizable product/brand entity found.
------------------------------------------------------------
Review 6: The Dell XPS 13 is sleek and fast, I'm very happy with my purchase.
 - No recognizable product/brand entity found.
------------------------------------------------------------
Review 7: Logitech mouse stopped working after just a week. Not worth the price.
 - Logitech (ORG)
------------------------------------------------------------
Review 8: Great performance from the ASUS ROG laptop - it runs all my games smoothly.
 - ASUS (ORG)
------------------------------------------------------------
Review 9: The Beats Studio3 have excellent bass and look stylish.
 - No recognizable product/brand entity found.
------------------------------------------------------------
Review 10: I regret buying the Lenovo tablet. It's very slow and laggy.
 - Lenovo (ORG)
------------------------------------------------------------
```

```
------------------------------------------------------------

❤️ Rule-Based Sentiment Analysis:

Review 1: I love my new Samsung Galaxy S21, the camera is amazing!
 - Polarity Score: 0.46 → Sentiment: Positive
------------------------------------------------------------
Review 2: The Apple AirPods Pro are so comfortable and the sound is top-notch.
 - Polarity Score: 0.60 → Sentiment: Positive
------------------------------------------------------------
Review 3: I was disappointed with the Sony headphones, poor noise cancellation.
 - Polarity Score: -0.57 → Sentiment: Negative
------------------------------------------------------------
Review 4: This HP laptop is very fast and reliable for work and school.
 - Polarity Score: 0.26 → Sentiment: Positive
------------------------------------------------------------
Review 5: Avoid the fake Ray-Ban sunglasses - cheap material and broke in 2 days!
 - Polarity Score: 0.00 → Sentiment: Neutral
------------------------------------------------------------
Review 6: The Dell XPS 13 is sleek and fast, I'm very happy with my purchase.
 - Polarity Score: 0.60 → Sentiment: Positive
------------------------------------------------------------
Review 7: Logitech mouse stopped working after just a week. Not worth the price.
 - Polarity Score: -0.15 → Sentiment: Negative
------------------------------------------------------------
Review 8: Great performance from the ASUS ROG laptop - it runs all my games smoothly.
 - Polarity Score: 0.60 → Sentiment: Positive
------------------------------------------------------------
Review 9: The Beats Studio3 have excellent bass and look stylish.
 - Polarity Score: 0.45 → Sentiment: Positive
------------------------------------------------------------
Review 10: I regret buying the Lenovo tablet. It's very slow and laggy.
 - Polarity Score: -0.39 → Sentiment: Negative
------------------------------------------------------------
```

```
Logistic Regression Accuracy: 1.0
Decision Tree Accuracy: 1.0
```



Bias Analysis in Our MNIST CNN Model and Mitigation Strategies Using TensorFlow Fairness Indicators and Rule-Based Systems

In our project, we developed a Convolutional Neural Network (CNN) to classify handwritten digits using the MNIST dataset. While MNIST is considered a well-balanced benchmark dataset, we conducted a critical assessment of our model to identify potential biases that could affect performance or fairness. Recognizing and mitigating these biases is essential, especially if the model were to be extended or deployed in more diverse, real-world scenarios.

## 🔍 Identifying Potential Biases

### 1. Class Imbalance Bias

Although MNIST is roughly balanced across all ten digits (0–9), our use of data augmentation through the ImageDataGenerator may inadvertently introduce skewed class distributions in specific mini-batches. This could cause the model to learn some digits more effectively than others, leading to uneven performance.

### 2. Augmentation-Induced Bias

Our data augmentation strategy involves random rotations, shifts, and zooming. While these techniques improve generalization, they may disproportionately affect digits with similar shapes. For instance, rotated versions of the digit "6" may resemble a "9", potentially confusing the model and resulting in classification bias against such digits.

### 3. Evaluation Bias

We initially relied on overall accuracy as the primary evaluation metric. However, this can be misleading if certain classes are harder to classify. To address this, we generated a detailed classification report and confusion matrix to assess per-class performance. This allowed us to observe that some digits (e.g., "5" and "8") had lower precision and recall scores compared to others like "1" or "0".

### 4. Confidence Bias

During prediction visualization, we noticed that the model tends to output significantly higher confidence scores for certain digits (such as "1" and "0"), while being less confident when predicting others. This confidence imbalance may lead to an over-reliance on "easier" digits and underrepresentation of more ambiguous ones.

## 🛠️ Mitigating Bias with TensorFlow Fairness Indicators (TFFI)

To further explore fairness, we proposed a synthetic subgrouping strategy using TensorFlow Fairness Indicators. Since MNIST does not contain sensitive attributes (e.g., gender, age), we created custom subgroups based on digit classification difficulty:

Group A (Simple Digits): 0, 1, 8

Group B (Complex Digits): 5, 6, 9

Using TensorFlow Model Analysis (TFMA) and Fairness Indicators, we can track metrics such as accuracy, precision, and recall across these groups. This helps determine whether the model performs equitably across simple and complex digits, ensuring that no group is consistently misclassified.

Furthermore, we plan to use these indicators to monitor performance gaps between subgroups and apply corrective strategies such as sample reweighting or targeted data augmentation if disparities are detected.

### 🧠 Rule-Based Strategies Inspired by spaCy

While spaCy is traditionally used in Natural Language Processing, we borrowed the concept of rule-based logic to flag edge cases in our image classification task. For instance, we introduced logic to highlight predictions where:

The predicted confidence is below a certain threshold (e.g., <60%)

The predicted class is commonly confused with another in the confusion matrix (e.g., “4” vs “9”)

This kind of post-processing helps us better understand and possibly correct high-risk predictions, thereby increasing transparency and accountability in our model's behavior.

### ✅ Summary

In summary, our analysis uncovered subtle but important forms of bias in our MNIST CNN model, including class, augmentation, and confidence-related imbalances. By integrating TensorFlow Fairness Indicators, we systematically evaluated fairness across custom-defined subgroups. In addition, we adopted rule-based diagnostics to catch potentially unreliable predictions.

These steps not only enhanced the robustness of our model but also aligned our project with ethical AI practices. Going forward, we intend to apply similar fairness evaluation techniques to more complex and real-world datasets beyond MNIST.

🧪 Buggy Code (Original Script)

python

Copy

Edit

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D
from tensorflow.keras.datasets import mnist

# Load dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize input
x_train = x_train / 255
x_test = x_test / 255

# Build model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28)),
    Flatten(),
    Dense(128, activation='relu'),
```

```
    Dense(10, activation='softmax')
])

# Compile with wrong loss function for one-hot labels
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train model (without converting labels to one-hot)
model.fit(x_train, y_train, epochs=5, batch_size=64)
```

✅ Corrected Code (Working Version)

python

Copy

Edit

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# Load dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize input
x_train = x_train / 255.0
x_test = x_test / 255.0

# FIXED: Add channel dimension (batch, 28, 28, 1)
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)
```

```
# FIXED: Convert labels to one-hot encoded
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Build model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')  # 10 classes for digits 0–9
])

# Compile model with correct loss
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train model
model.fit(x_train, y_train, epochs=5, batch_size=64, validation_data=(x_test, y_test))
```

🛠️ Bug Fix Summary

As a group, we reviewed a buggy TensorFlow MNIST classifier and identified multiple issues:

Shape Mismatch in Input: The input to Conv2D was (28, 28) but should have been (28, 28, 1) since MNIST images are grayscale. We fixed this by reshaping the training data with .reshape(-1, 28, 28, 1).

Incorrect Label Format: The model used categorical_crossentropy as the loss function, which expects one-hot encoded labels. However, the labels were raw integers. We corrected this using to_categorical() from Keras.

No Validation Data: We also added validation_data=(x_test, y_test) to monitor model performance during training.

These corrections ensured the model ran smoothly and achieved high accuracy on MNIST digit classification