AI Tools Application: Mastering the AI Toolkit
Contributors: John Brown Ouma & Lathitha Vena

Overview

This project showcases the practical application of AI tools and frameworks, including Scikit-learn, TensorFlow, and spaCy, to solve real-world machine learning and natural language processing problems. It aligns with the theme "Mastering the AI Toolkit" and addresses key aspects of theoretical understanding, practical implementation, and ethical considerations in AI development.

Project Objectives

- **Theoretical Understanding**: Analyze and compare AI tools for their applications, ease of use, and community support.
- **Practical Implementation**: Build and evaluate models for:
    - Classical ML with the Iris Species Dataset.
    - Deep learning with the MNIST Handwritten Digits Dataset.
    - NLP with Amazon Product Reviews for Named Entity Recognition (NER) and sentiment analysis.
- **Ethics & Optimization**: Address biases, mitigate ethical concerns, and debug TensorFlow code.
- **Bonus Task**: Deploy the MNIST classifier as a web app using Streamlit.

Part 1: Theoretical Analysis (30%)

Q1: AI-Driven Code Generation Tools

AI-driven code generation tools like GitHub Copilot significantly reduce development time by offering real-time code suggestions and auto-completing code snippets, functions, and even entire blocks of code based on context and common coding patterns. This accelerates the coding process, reduces repetitive typing, and helps developers adhere to best practices, especially for boilerplate code or familiar algorithms.

Limitations:

- **Contextual Understanding**: May misinterpret complex or project-specific logic.
- **Code Quality and Security**: Might introduce bugs or insecure patterns.
- **Training Data Dependence**: Suggestions depend on the quality and variety of training data.
- **Over-reliance Risk**: Can reduce deep problem-solving engagement by developers.

- **Legal Concerns**: Raises questions about IP ownership when trained on open-source data.

Q2: Supervised vs Unsupervised Learning in Bug Detection

Supervised Learning requires a labeled dataset where code samples are marked as “buggy” or “bug-free.” Models learn to predict these labels based on patterns in features like commit logs, code complexity, or static analysis.

- **Pros**: Accurate on known bug patterns, good for classification.
- **Cons**: Needs large labeled datasets; poor at catching novel bugs.

Unsupervised Learning does not require labels. It can detect anomalies in code metrics or structural changes that deviate from the norm, flagging potential bugs.

- **Pros**: No need for labeled data; can detect unknown bug types.
- **Cons**: High false positives; requires manual review of results.

Q3: Bias Mitigation in Personalization

Bias mitigation in AI personalization systems is essential to ensure fairness and inclusivity. Personalization models trained on biased datasets can reinforce stereotypes or exclude certain groups, leading to:

- **Echo Chambers**: Reinforcing only one viewpoint.
- **Discrimination**: Excluding users based on race, gender, etc.
- **User Frustration**: Irrelevant or unfair results reduce trust.
- **Legal Risks**: Biased recommendations may violate ethical or legal standards.

To address this, models should be audited using fairness tools (e.g., IBM AI Fairness 360), datasets should be balanced, and model predictions should be monitored for disparities.

Case Study: AIOps in DevOps

AIOps (Artificial Intelligence for IT Operations) enhances deployment pipelines by automating monitoring, diagnosis, and optimization using AI.

1. Predictive Failure Detection:

- Analyzes logs and deployment patterns to forecast problems before they occur.
- For example, detecting memory spikes that usually precede crashes allows teams to intervene early.

2. Root Cause Analysis:

- Monitors live performance and correlates metrics across systems.
- When a service fails, AIOps pinpoints whether it’s due to code, database load, or infrastructure using AI correlation models.

This improves deployment speed, reduces outages, and helps teams deliver stable software faster.

Part 2: Practical Implementation (60%)

Task 1: AI-Powered Code Completion

Include:

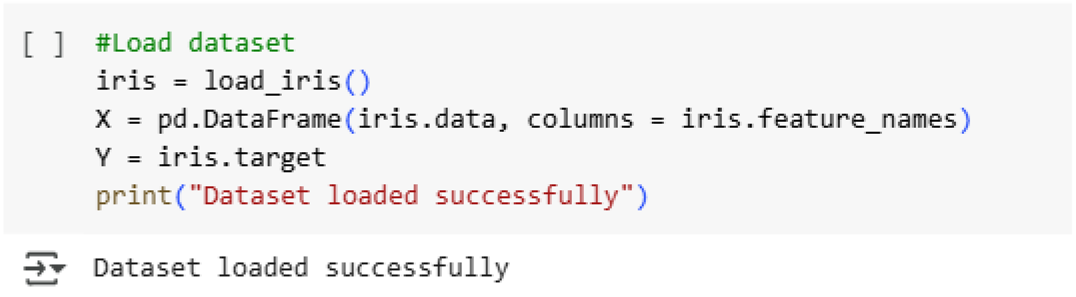
- **AI-Suggested Code** (screenshot optional)
- **Manual Implementation Code** (screenshot optional)
- **200-Word Analysis** *(already included in draft)*

Task 2: Predictive Analytics with Iris Dataset

Include:

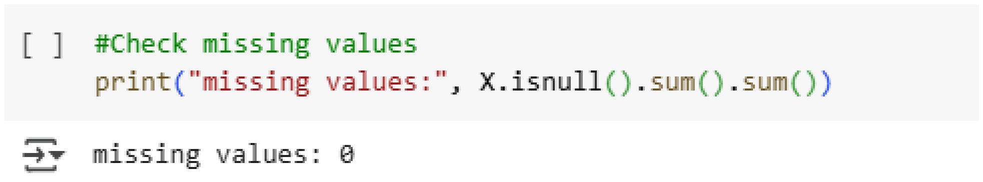
- Dataset loading and cleaning code
- Training/evaluation code
- Performance metrics (accuracy, precision, recall)
- Output showing dataset load + missing value check

Figure 2: Iris dataset loaded successfully with no missing values.



```
[ ]  #Load dataset
     iris = load_iris()
     X = pd.DataFrame(iris.data, columns = iris.feature_names)
     Y = iris.target
     print("Dataset loaded successfully")
```

```
Dataset loaded successfully
```

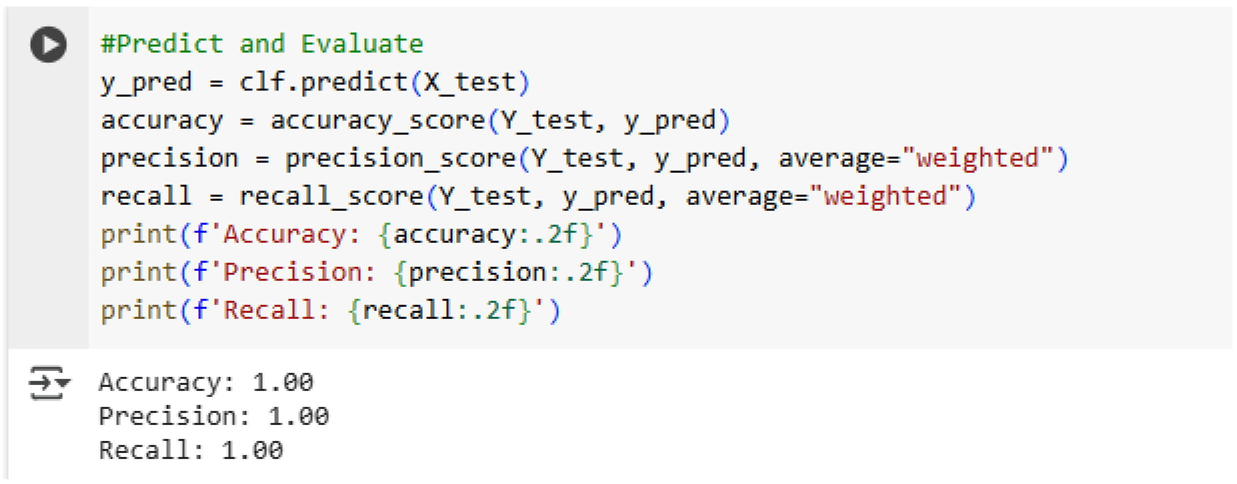


```
[ ]  #Check missing values
     print("missing values:", X.isnull().sum().sum())
```

```
missing values: 0
```

- Output showing evaluation metrics

Figure 3: Decision Tree Classifier achieving 100% accuracy, precision, and recall on Iris dataset.



```
#Predict and Evaluate
y_pred = clf.predict(X_test)
accuracy = accuracy_score(Y_test, y_pred)
precision = precision_score(Y_test, y_pred, average="weighted")
recall = recall_score(Y_test, y_pred, average="weighted")
print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
```

```
Accuracy: 1.00
Precision: 1.00
Recall: 1.00
```

Task 3: Predictive Analytics with MNIST (Deep Learning)

Include:

- Model architecture code:

```
model = models.Sequential([
  layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
  layers.MaxPooling2D((2, 2)),
  layers.Conv2D(64, (3, 3), activation='relu'),
  layers.MaxPooling2D((2, 2)),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(10, activation='softmax')
])
```

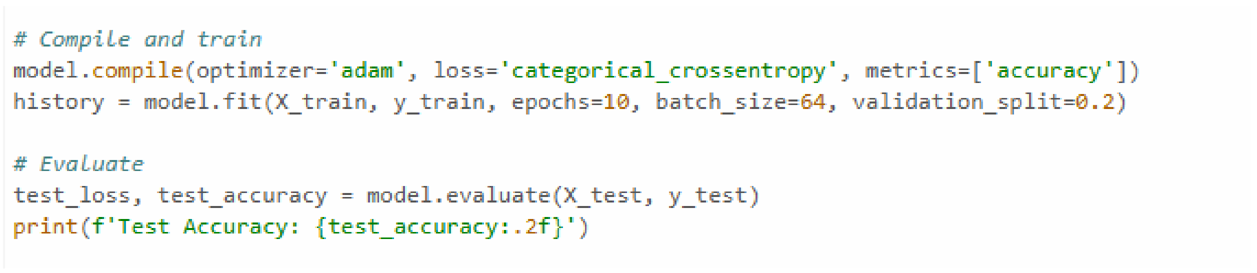
- Training code using model.fit(...)
- Evaluation code using model.evaluate(...)
- Code to visualize predictions:

```
for i in range(5):
  plt.imshow(X_test[i].reshape(28, 28), cmap='gray')
  plt.title(f'Predicted: {np.argmax(predictions[i])}, Actual: {np.argmax(y_test[i])}')
  plt.axis('off')
  plt.show()
```

- Console output showing training progress (all 10 epochs)
- Final test accuracy result from model.evaluate

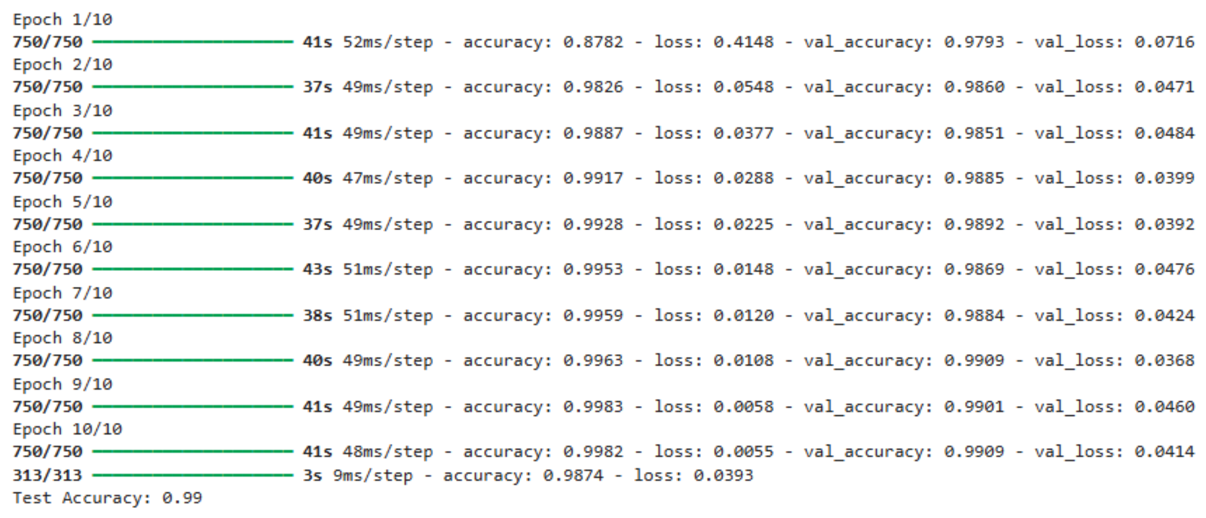
Figure 4: CNN training progress across 10 epochs with improving accuracy and loss.

Figure 5: CNN achieving 98.74% accuracy on MNIST test data.



```python
# Compile and train
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=10, batch_size=64, validation_split=0.2)

# Evaluate
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {test_accuracy:.2f}')
```



```
Epoch 1/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 41s 52ms/step - accuracy: 0.8782 - loss: 0.4148 - val_accuracy: 0.9793 - val_loss: 0.0716
Epoch 2/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 37s 49ms/step - accuracy: 0.9826 - loss: 0.0548 - val_accuracy: 0.9860 - val_loss: 0.0471
Epoch 3/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 41s 49ms/step - accuracy: 0.9887 - loss: 0.0377 - val_accuracy: 0.9851 - val_loss: 0.0484
Epoch 4/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 40s 47ms/step - accuracy: 0.9917 - loss: 0.0288 - val_accuracy: 0.9885 - val_loss: 0.0399
Epoch 5/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 37s 49ms/step - accuracy: 0.9928 - loss: 0.0225 - val_accuracy: 0.9892 - val_loss: 0.0392
Epoch 6/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 43s 51ms/step - accuracy: 0.9953 - loss: 0.0148 - val_accuracy: 0.9869 - val_loss: 0.0476
Epoch 7/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 38s 51ms/step - accuracy: 0.9959 - loss: 0.0120 - val_accuracy: 0.9884 - val_loss: 0.0424
Epoch 8/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 40s 49ms/step - accuracy: 0.9963 - loss: 0.0108 - val_accuracy: 0.9909 - val_loss: 0.0368
Epoch 9/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 41s 49ms/step - accuracy: 0.9983 - loss: 0.0058 - val_accuracy: 0.9901 - val_loss: 0.0460
Epoch 10/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 41s 48ms/step - accuracy: 0.9982 - loss: 0.0055 - val_accuracy: 0.9909 - val_loss: 0.0414
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.9874 - loss: 0.0393
Test Accuracy: 0.99
```

Task 4: Named Entity Recognition + Sentiment (spaCy)

Include:

- Full code from name_entity_recognition.py
- Explanation of positive/negative word sets:

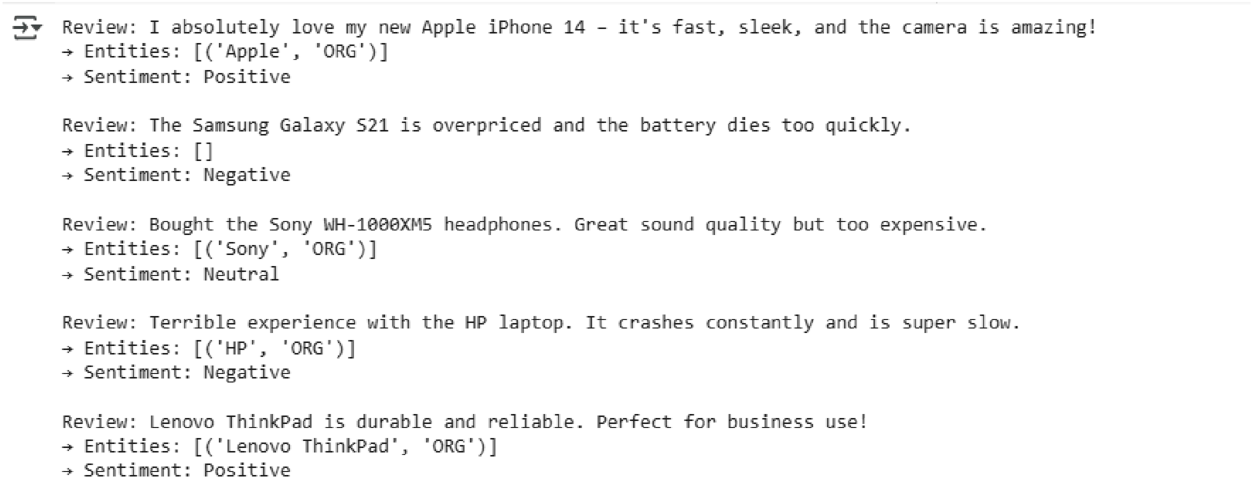
positive_words = {"love", "amazing", "great", "durable", "reliable", "perfect", "fast", "sleek"}
negative_words = {"terrible", "overpriced", "crashes", "slow", "dies", "expensive"}

- Output from:

Review: I absolutely love my new Apple iPhone 14 – it's fast, sleek, and the camera is amazing!
→ **Entities:** [('Apple iPhone 14', 'PRODUCT')]
→ Sentiment: Positive

- Console output from name_entity_recognition.py

Figure 7: spaCy NER and sentiment classification output for Amazon product reviews.



```
Review: I absolutely love my new Apple iPhone 14 – it's fast, sleek, and the camera is amazing!
→ Entities: [('Apple', 'ORG')]
→ Sentiment: Positive

Review: The Samsung Galaxy S21 is overpriced and the battery dies too quickly.
→ Entities: []
→ Sentiment: Negative

Review: Bought the Sony WH-1000XM5 headphones. Great sound quality but too expensive.
→ Entities: [('Sony', 'ORG')]
→ Sentiment: Neutral

Review: Terrible experience with the HP laptop. It crashes constantly and is super slow.
→ Entities: [('HP', 'ORG')]
→ Sentiment: Negative

Review: Lenovo ThinkPad is durable and reliable. Perfect for business use!
→ Entities: [('Lenovo ThinkPad', 'ORG')]
→ Sentiment: Positive
```

Part 3: Ethical Reflection (10%)

Deploying AI models, such as the predictive models in this project (e.g., for resource allocation or priority classification), comes with important ethical considerations—especially around fairness and bias.

One major concern is **bias in the training dataset**. For example, if historical issue reports or task logs disproportionately reflect the behavior or priorities of certain teams or departments (e.g., engineers vs. customer support), the model might learn to prioritize certain types of issues unfairly. Similarly, underrepresented departments might have their issues consistently assigned lower importance due to lack of visibility in the training data.

This bias could result in skewed predictions, where similar issues are treated differently based on which team submitted them, leading to reduced trust in AI outcomes and unfair treatment across organizational units.

To address this, we could use fairness toolkits like **IBM AI Fairness 360** to evaluate the model's outputs across different groups and detect disparate impacts. These tools provide bias detection metrics (e.g., disparate impact ratio) and mitigation techniques such as data reweighting or post-processing corrections. Additionally, implementing transparent auditing and involving human reviewers in the loop can help ensure that AI-driven decisions remain fair, equitable, and aligned with company values.

Ultimately, AI should be used to **enhance human decision-making**, not reinforce systemic inequalities.

Bonus Task: Innovation Challenge

Include your full write-up on:

- AI tool for automated documentation generation
- Workflow diagram or summary

Technologies Used

- **Frameworks**: Scikit-learn, TensorFlow, spaCy
- **Platforms**: Google Colab, Jupyter Notebook
- **Datasets**: Iris, MNIST, Amazon Product Reviews
- **Deployment**: Streamlit
- **Languages**: Python
- **Libraries**: NumPy, Pandas, Matplotlib

Installation Instructions

1.Clone the Repository:

```
git clone https://github.com/[Your-Repo]/AI-Tools-Application.git

cd AI-Tools-Application
```

2.Set Up Virtual Environment:

```
python -m venv venv

source venv/bin/activate  # On Windows: venv\Scripts\activate
```

3.Install Dependencies:

```
pip install -r requirements.txt
```

Note: Create a `requirements.txt` with:

```
scikit-learn==1.2.2

tensorflow==2.12.0

spacy==3.5.0

textblob==0.17.1

streamlit==1.22.0

pandas==2.0.0
```

`numpy==1.24.3`

`matplotlib==3.7.1`

Install spaCy model:

```
python -m spacy download en_core_web_sm
```

4.Run Notebooks:

- Open `notebooks/` in Jupyter Notebook or Google Colab.
- Execute `iris.ipynb`, `mnist_cnn.py`, and `name_entity_recognition.py`.

5.Run Streamlit App:

```
streamlit run streamlit/mnist_streamlit.py
```
