Comparative Evaluation of Machine Learning and Deep Learning Models for Binary Classification

Ⅰ) Introduction:

In this report, we evaluate the performance of three machine learning (ML) models and three deep learning (DL) models for a binary classification task. We will work on the **mushroom dataset**, which has been cleaned and is balanced, ensuring that each class is equally represented. The goal is to compare the effectiveness of traditional machine learning approaches, such as Logistic Regression, Decision Trees, and Random Forests, with modern deep learning techniques, including Artificial Neural Networks (ANN), Convolutional Neural Networks (CNN), and Recurrent Neural Networks (RNN). Each model has unique strengths, with ML models typically being simpler and more interpretable, while DL models are capable of learning more complex patterns and interactions in the data. Through this analysis, we aim to determine which models perform best in terms of accuracy and F1-score.

Ⅱ) Machine Learning Models:

1)LogisticRegression:

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train the model on the scaled data
model = LogisticRegression()
model.fit(X_train_scaled, y_train)

# Make predictions and calculate accuracy
y_pred = model.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
f1 = f1_score(y_test, y_pred)
print(f"F1-Score: {f1:.2f}")
```

```
Accuracy: 0.64
F1-Score: 0.68
```

Interpretation:
The accuracy of 0.64 means the Logistic Regression model correctly classified 64% of the test samples. While this is reasonable, it suggests that the model may not be capturing all the patterns in the data.

The F1-score of 0.68, which is higher than the accuracy, indicates that the model is performing better in terms of balancing precision and recall, even though the dataset is balanced.

2)Decision Tree:

```
model = DecisionTreeClassifier()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
f1 = f1_score(y_test, y_pred)
print(f"F1-Score: {f1:.2f}")
```

```
Accuracy: 0.98
F1-Score: 0.98
```

Interpretation:
The **accuracy** of 0.98 means that the Decision Tree model correctly classified 98% of the test samples, which is an excellent result. This suggests that the model is effectively capturing the patterns in the balanced dataset.

The **F1-score** of 0.98, being close to the accuracy, further confirms that the model is performing very well in balancing precision and recall. The high F1-score indicates that both false positives and false negatives are minimized, making the model reliable for both classes.

3)Random Forest:

```
model = RandomForestClassifier(n_estimators=200, max_depth=15, random_state=42)
model.fit(X_train, y_train)

# Evaluate performance
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy with n_estimators=200 and max_depth=15: {accuracy:.2f}")
f1 = f1_score(y_test, y_pred)
print(f"F1-Score with n_estimators=200 and max_depth=15: {f1:.2f}")
```

```
Accuracy with n_estimators=200 and max_depth=15: 0.99
F1-Score with n_estimators=200 and max_depth=15: 0.99
```

Interpretation:
The **accuracy** of 0.99 demonstrates that the Random Forest model is highly effective, correctly classifying 99% of the test samples. This result shows that the ensemble approach of combining multiple decision trees is powerful for this binary classification task.The **F1-score** of 0.99 matches the accuracy, indicating that the model is excelling in both precision and recall. This confirms that the Random Forest model minimizes false positives and false negatives, ensuring balanced performance for both classes.

Conclusion

- Logistic Regression: Accuracy of 64% and F1-score of 0.68, providing a simple baseline but with limited predictive power.
- Decision Tree: Accuracy and F1-score of 98%, showing strong performance with its ability to handle complex patterns.
- Random Forest: Accuracy and F1-score of 99%, outperforming the others due to its ensemble learning approach.

Overall, Random Forest demonstrated the best performance, making it the most effective algorithm for this task.

Ⅱ Ⅰ) Deep Learning Models:

1)Artificial Neural Network (ANN):

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(16, input_shape=(X_train.shape[1],), activation='relu'),
    tf.keras.layers.Dense(8, activation='relu'),  # Second hidden layer with 8 neuron
    tf.keras.layers.Dense(1, activation='sigmoid')  # Single neuron with sigmoid activation
])

# Compile the model for binary classification
model.compile(optimizer='adam',
              loss='binary_crossentropy',  # Binary cross-entropy for binary classification
              metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=1)
print(f'Test accuracy: {test_accuracy}')
```

The **Artificial Neural Network (ANN)** was evaluated on the balanced mushroom dataset for binary classification. The model was built using **two hidden layers**: the first with **16 neurons** and the second with **8 neurons**, both utilizing the **ReLU activation function**. The output layer consisted of a **single neuron** with a **sigmoid activation function** to handle binary classification.The model was compiled with the **Adam optimizer** and **binary cross-entropy loss function**, and trained over **50 epochs** with a **batch size of 32**, ensuring effective learning. After training, the ANN achieved a **test accuracy of 66.08%**.

Interpretation:

The accuracy of **75.08%** indicates that the ANN effectively captured patterns in the dataset, demonstrating its ability to model relationships between the input features and the target variable. The chosen architecture and hyperparameters contributed to this performance, highlighting the model's ability to process structured data effectively.

2) Recurrent Neural Network (RNN):

```
X_train_rnn = X_train.values.reshape(X_train.shape[0], X_train.shape[1], 1)  # Adding 1 for single timestep
X_test_rnn = X_test.values.reshape(X_test.shape[0], X_test.shape[1], 1)  # Adding 1 for single timestep

model = tf.keras.Sequential([
    tf.keras.layers.LSTM(16, input_shape=(X_train_rnn.shape[1], X_train_rnn.shape[2]), activation='relu', return_sequences=False),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(X_train_rnn, y_train, epochs=30, batch_size=32, validation_data=(X_test_rnn, y_test))

test_loss, test_accuracy = model.evaluate(X_test_rnn, y_test, verbose=1)
print(f'Test accuracy: {test_accuracy:.2f}')
```

The model included:

- An LSTM layer with 16 units to capture patterns in the data.
- A Dense hidden layer with 8 neurons (ReLU activation).
- A Dense output layer with 1 neuron (sigmoid activation).

The model was trained for 30 epochs with a batch size of 32 and achieved a test accuracy of 97%.

Interpretation:
The RNN demonstrated excellent performance with a 97% accuracy, effectively learning the patterns in the dataset. The LSTM layer contributed to its ability to handle structured input data, even without sequential dependencies.

3) Convolutional Neural Network(CNN):

```
# Reshape the input data to 3D for CNN (batch_size, features, 1)
X_train_cnn = X_train.values.reshape(X_train.shape[0], X_train.shape[1], 1)  # Reshaping to 3D
X_test_cnn = X_test.values.reshape(X_test.shape[0], X_test.shape[1], 1)  # Reshaping to 3D

# Build the CNN model for binary classification
model = tf.keras.Sequential([
    tf.keras.layers.Conv1D(32, kernel_size=3, activation='relu', padding='same', input_shape=(X_train_cnn.shape[1], 1)),  # 1D convolutional layer with padding
    tf.keras.layers.MaxPooling1D(pool_size=2),  # Max pooling layer to reduce dimensionality
    tf.keras.layers.Conv1D(64, kernel_size=3, activation='relu', padding='same'),  # Second convolutional layer with padding
    tf.keras.layers.MaxPooling1D(pool_size=2),  # Another max pooling layer
    tf.keras.layers.Flatten(),  # Flatten the output to feed into the fully connected layers
    tf.keras.layers.Dense(32, activation='relu'),  # Dense hidden layer
    tf.keras.layers.Dense(1, activation='sigmoid')  # Output layer with sigmoid for binary classification
])
model.compile(optimizer='adam',
              loss='binary_crossentropy',  # Binary cross-entropy for binary classification
              metrics=['accuracy'])

model.fit(X_train_cnn, y_train, epochs=50, batch_size=32, validation_data=(X_test_cnn, y_test))

test_loss, test_accuracy = model.evaluate(X_test_cnn, y_test, verbose=1)
print(f'Test accuracy: {test_accuracy:.2f}')
```

The model was trained for 50 epochs with a batch size of 32 and achieved a test accuracy of 95%.

Interpretation:
The CNN achieved a strong performance with 95% accuracy, effectively learning patterns from the input features. The convolutional layers captured spatial dependencies, while pooling layers reduced the computational complexity, contributing to the model's efficiency and effectiveness.

Conclusion:

In this study, we evaluated the performance of three machine learning (ML) algorithms—Logistic Regression, Decision Tree, and Random Forest—along with three deep learning (DL) models—Artificial Neural Network (ANN), Recurrent Neural Network (RNN), and Convolutional Neural Network (CNN)—for a binary classification task using the balanced mushroom dataset.

- **Machine Learning:**
    - Logistic Regression provided a solid baseline with an accuracy of 64% and an F1-score of 68%. While its performance was decent, it highlighted the simplicity of the model and its potential limitations in capturing complex patterns.
    - Decision Tree demonstrated excellent performance with an accuracy and F1-score of 98%, showcasing its ability to model complex decision boundaries effectively.
    - Random Forest further improved on this, achieving an accuracy and F1-score of 99%, highlighting its robustness and capacity for handling more complex datasets by leveraging multiple decision trees.
- **Deep Learning:**
    - ANN achieved a test accuracy of 66%, demonstrating its ability to model non-linear relationships and capture patterns in the data. However, it showed room for improvement compared to the tree-based models.
    - RNN excelled with a test accuracy of 97%, showing its potential even with structured, non-sequential data. The LSTM layer helped the model learn dependencies and relationships effectively.
    - CNN performed slightly lower than the RNN with a test accuracy of 95%, yet still showed remarkable ability to extract features from the input data. The convolutional layers helped capture local patterns, though not as effectively as the RNN's temporal learning.

Overall, the tree-based algorithms (Decision Tree and Random Forest) performed exceptionally well, with Random Forest providing the best results. Deep learning models, while performing admirably, had slightly lower accuracies, with RNN showing the best performance among them. However, it is important to note that the results are highly dependent on the specific nature of the problem, which in this case is a binary classification task. Deep learning algorithms like RNN and CNN are generally more powerful for tasks involving complex relationships and large datasets, but for simpler binary classification problems, traditional machine learning models, especially Random Forest, may yield better performance.