Uncertainty in AI :

Implement Inferencing with Bayesian Network in Python

[6]

```python
from pgmpy.models import BayesianNetwork
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination
```

```python
# Define the Bayesian Network structure
model = BayesianNetwork([('Cloudy', 'Sprinkler'),
                         ('Cloudy', 'Rain'),
                         ('Sprinkler', 'WetGrass'),
                         ('Rain', 'WetGrass')])
```

[8]

```python
# Define Conditional Probability Distributions (CPDs)
# CPD for Cloudy
cpd_cloudy = TabularCPD(variable='Cloudy', variable_card=2, values=[[0.5], [0.5]])

# CPD for Sprinkler given Cloudy
cpd_sprinkler = TabularCPD(variable='Sprinkler', variable_card=2,
                           values=[[0.5, 0.9], [0.5, 0.1]],
                           evidence=['Cloudy'], evidence_card=[2])

# CPD for Rain given Cloudy
cpd_rain = TabularCPD(variable='Rain', variable_card=2,
                      values=[[0.8, 0.2], [0.2, 0.8]],
```

```python
# CPD for WetGrass given Sprinkler and Rain
cpd_wet_grass = TabularCPD(variable='WetGrass', variable_card=2,
                           values=[[1.0, 0.1, 0.1, 0.01], [0.0, 0.9, 0.9, 0.99]],
                           evidence=['Sprinkler', 'Rain'], evidence_card=[2, 2])

# Add CPDs to the model
model.add_cpds(cpd_cloudy, cpd_sprinkler, cpd_rain, cpd_wet_grass)

# Check if the model is valid
print("Is model valid? ", model.check_model())
```

```
Is model valid?  True
```

```python
# Perform inference using Variable Elimination
inference = VariableElimination(model)

# Query the probability of WetGrass being wet, given that it is cloudy
result = inference.query(variables=['WetGrass'], evidence={'Cloudy': 1})
print(result)
```

```
+-------------+-----------------+
| WetGrass    |   phi(WetGrass) |
+=============+=================+
| WetGrass(0) |          0.2548 |
+-------------+-----------------+
| WetGrass(1) |          0.7452 |
+-------------+-----------------+
```

Cognitive Computing :

Building a Cognitive Healthcare Application

- Patient Data Analysis: Use AI to analyze patient records and medical history.
- Predictive Analytics: Predict health risks and outcomes based on data patterns.
- Personalized Treatment: Tailor treatments to individual patients using cognitive insights.
- Virtual Health Assistants: Implement chatbots for patient inquiries and appointment scheduling.
- Remote Monitoring: Monitor patient health through wearable devices and AI algorithms.

Smarter Cities: Cognitive Computing in Government

- **Traffic Management**: Optimize traffic flow and reduce congestion using real-time data.
- **Public Safety**: Analyze data for crime patterns and deploy resources effectively.
- **Waste Management**: Use sensors to optimize garbage collection routes.
- **Citizen Engagement**: Implement chatbots for citizen queries and service requests.
- **Energy Management**: Monitor and optimize energy consumption in public buildings..

Cognitive Computing in Insurance

- **Fraud Detection**: Analyze claims data to identify fraudulent activities.
- **Risk Assessment**: Evaluate risks using big data and predictive modeling.
- **Customer Insights**: Personalize insurance products based on customer behavior.
- **Claims Processing**: Automate and speed up claims processing with AI.
- **Chatbots**: Provide 24/7 customer support for inquiries and claims.

Cognitive Computing in Customer Service

- **Automated Support**: Use AI chatbots to handle common customer queries.
- **Sentiment Analysis**: Analyze customer feedback to improve services.
- **Personalization**: Tailor customer interactions based on previous behavior.
- **Predictive Support**: Anticipate customer needs and offer proactive solutions.
- **Data Analytics**: Use customer data to optimize service strategies and operations.

Fuzzy Logic & Its Applications :

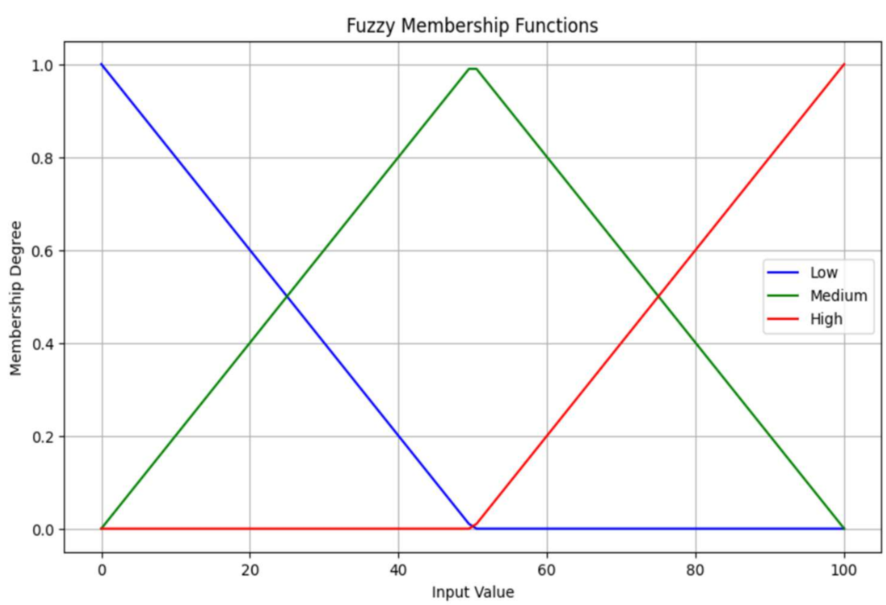
a) Implementation of Fuzzy Membership Functions :

```python
# Implementation of Fuzzy Membership Functions
import numpy as np
import matplotlib.pyplot as plt
import skfuzzy as fuzz

# Generate a range of values
x = np.linspace(0, 100, 100)

# Define fuzzy membership functions
low = fuzz.trimf(x, [0, 0, 50])
medium = fuzz.trimf(x, [0, 50, 100])
high = fuzz.trimf(x, [50, 100, 100])

# Plot the membership functions
plt.figure(figsize=(10, 6))
plt.plot(x, low, 'b', label='Low')
plt.plot(x, medium, 'g', label='Medium')
plt.plot(x, high, 'r', label='High')
plt.title('Fuzzy Membership Functions')
plt.xlabel('Input Value')
plt.ylabel('Membership Degree')
plt.legend()
plt.grid()
plt.show()
```

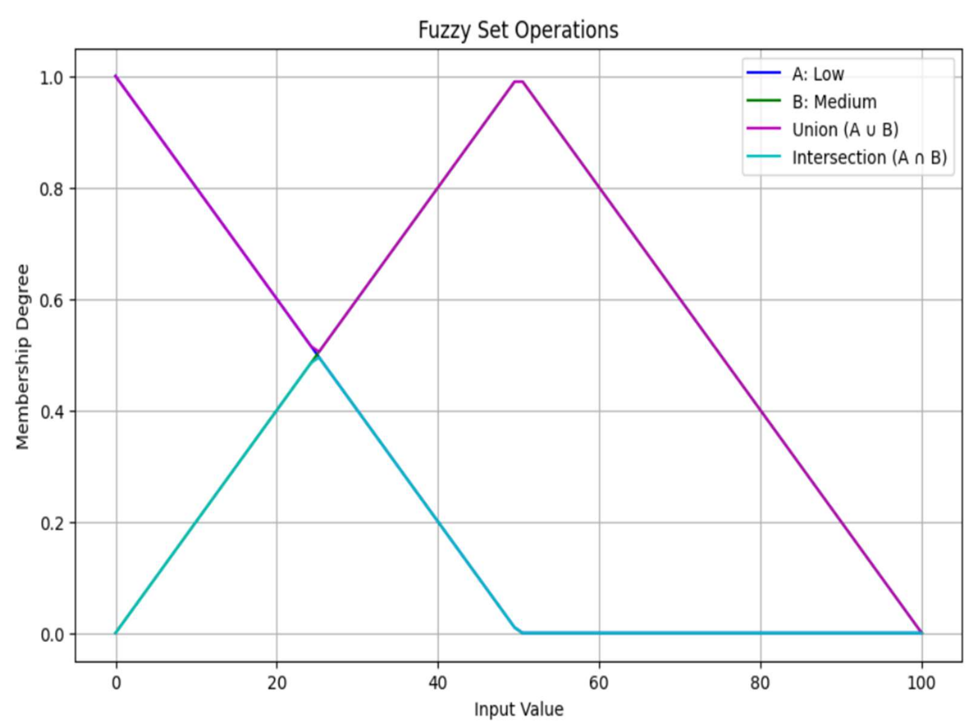


b) Implementation of fuzzy set Properties :

```
[5] # Implementation of Fuzzy Set Properties
    # Define fuzzy sets
    A = low
    B = medium

    # Union and Intersection
    union = np.maximum(A, B)
    intersection = np.minimum(A, B)

    # Plot the results
    plt.figure(figsize=(10, 6))
    plt.plot(x, A, 'b', label='A: Low')
    plt.plot(x, B, 'g', label='B: Medium')
    plt.plot(x, union, 'm', label='Union (A ∪ B)')
    plt.plot(x, intersection, 'c', label='Intersection (A ∩ B)')
    plt.title('Fuzzy Set Operations')
    plt.xlabel('Input Value')
    plt.ylabel('Membership Degree')
    plt.legend()
    plt.grid()
    plt.show()
```



c) Design of a Fuzzy control system using Fuzzy tool :

```python
import numpy as np
import skfuzzy as fuzz

# Define the fuzzy variables and their membership functions
temperature = np.linspace(0, 100, 100)
fan_speed = np.linspace(0, 100, 100)

# Membership functions for temperature
cold = fuzz.trimf(temperature, [0, 0, 50])
comfortable = fuzz.trimf(temperature, [0, 50, 100])
hot = fuzz.trimf(temperature, [50, 100, 100])

# Membership functions for fan speed
low_speed = fuzz.trimf(fan_speed, [0, 0, 50])
medium_speed = fuzz.trimf(fan_speed, [0, 50, 100])
high_speed = fuzz.trimf(fan_speed, [50, 100, 100])

# Fuzzification (example input)
input_temp = 75
cold_level = fuzz.interp_membership(temperature, cold, input_temp)
comfortable_level = fuzz.interp_membership(temperature, comfortable, input_temp)
hot_level = fuzz.interp_membership(temperature, hot, input_temp)

# Initialize output membership degrees
fan_speed_low = cold_level  # Low speed if the temperature is cold
fan_speed_medium = comfortable_level  # Medium speed if the temperature is comfortable
fan_speed_high = hot_level  # High speed if the temperature is hot
```

```python
fan_speed_medium = comfortable_level  # Medium speed if the temperature is comfortable
fan_speed_high = hot_level  # High speed if the temperature is hot

# Step 5: Aggregate the output membership functions
aggregated_fan_speed = np.fmax(np.fmax(low_speed * fan_speed_low, medium_speed * fan_speed_medium), high_speed * fan_speed_high)

# Defuzzification (Crisp Output)
defuzzified_output = fuzz.defuzz(fan_speed, aggregated_fan_speed, 'centroid')

print(f'Fuzzified Fan Speed Output: {defuzzified_output:.2f}%')
```

```
Fuzzified Fan Speed Output: 58.34%
```

Introduction to Deep Learning :

- Handwritten Digit Recognition System (like MNIST Dataset)

```python
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

```python
# Load the MNIST dataset
(train_images, train_labels), (test_images, test_labels) = datasets.mnist.load_data()

# Normalize the pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0

# Reshape the images to add a channel dimension (28x28x1)
train_images = train_images.reshape((train_images.shape[0], 28, 28, 1))
test_images = test_images.reshape((test_images.shape[0], 28, 28, 1))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```python
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(64, activation='relu'),
    layers.Dense(10, activation='softmax')  # 10 classes for digits 0-9
])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_di
```

```
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```python
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```python
model.fit(train_images, train_labels, epochs=5, validation_data=(test_images, test_labels))
```

```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 71s 37ms/step - accuracy: 0.9061 - loss: 0.3175 - val_accuracy: 0.9808 - val_loss: 0.0587
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 78s 35ms/step - accuracy: 0.9846 - loss: 0.0491 - val_accuracy: 0.9885 - val_loss: 0.0344
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 86s 37ms/step - accuracy: 0.9889 - loss: 0.0338 - val_accuracy: 0.9903 - val_loss: 0.0313
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 81s 37ms/step - accuracy: 0.9928 - loss: 0.0229 - val_accuracy: 0.9886 - val_loss: 0.0356
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 81s 36ms/step - accuracy: 0.9943 - loss: 0.0183 - val_accuracy: 0.9894 - val_loss: 0.0351
<keras.src.callbacks.history.History at 0x795e8cf3d6f0>
```

```python
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_acc}')
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.9857 - loss: 0.0455
Test accuracy: 0.9894000291824341
```

```
Test accuracy: 0.9894000291824341
```

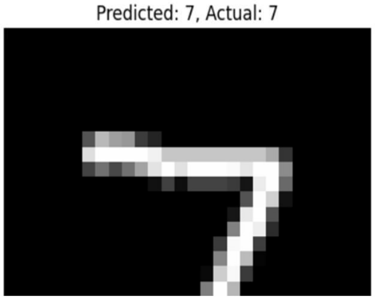
```python
predictions = model.predict(test_images)

# Example: Print the prediction for the first image
print(f'Predicted label: {predictions[0].argmax()}')
print(f'Actual label: {test_labels[0]}')
```

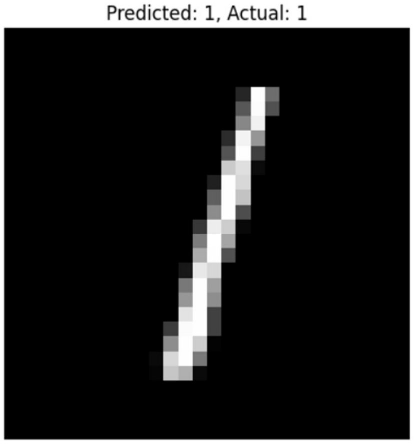
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step
Predicted label: 7
Actual label: 7
```

```python
# Plot the first 5 test images and their predicted labels
for i in range(5):
    plt.imshow(test_images[i].reshape(28, 28), cmap='gray')
    plt.title(f'Predicted: {predictions[i].argmax()}, Actual: {test_labels[i]}')
    plt.axis('off')
    plt.show()
```

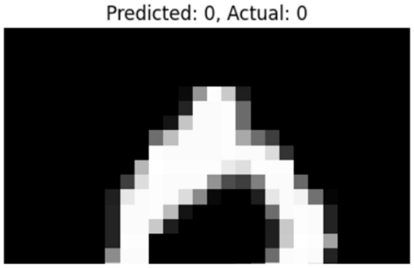
Predicted: 7, Actual: 7



Predicted: 1, Actual: 1



Predicted: 0, Actual: 0



Implementation of supervised learning algorithm like

a. Ada-Boosting

b. Random forests

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
import seaborn as sns
```

```
[ ] # Load the Iris dataset
data = load_iris()
X = data.data  # Features
y = data.target  # Labels

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
[ ] # Initialize Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
rf_model.fit(X_train, y_train)

# Predict on the test set
y_pred_rf = rf_model.predict(X_test)

# Evaluate the model
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
```

```
Random Forest Accuracy: 1.0
```

```
[ ] # Initialize AdaBoost Classifier
adaboost_model = AdaBoostClassifier(n_estimators=100, random_state=42)

# Train the model
adaboost_model.fit(X_train, y_train)

# Predict on the test set
y_pred_adaboost = adaboost_model.predict(X_test)

# Evaluate the model
print("AdaBoost Accuracy:", accuracy_score(y_test, y_pred_adaboost))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default) is deprecate
  warnings.warn(
AdaBoost Accuracy: 1.0
```

```python
# Cross-validation for Random Forest
rf_cv_scores = cross_val_score(rf_model, X, y, cv=5)
print("Random Forest Cross-Validation Scores:", rf_cv_scores)
print("Average CV Score (Random Forest):", np.mean(rf_cv_scores))

# Cross-validation for AdaBoost
adaboost_cv_scores = cross_val_score(adaboost_model, X, y, cv=5)
print("AdaBoost Cross-Validation Scores:", adaboost_cv_scores)
print("Average CV Score (AdaBoost):", np.mean(adaboost_cv_scores))
```

```
Random Forest Cross-Validation Scores: [0.96666667 0.96666667 0.93333333 0.96666667 1.        ]
Average CV Score (Random Forest): 0.9666666666666668
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default) is deprecate
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default) is deprecate
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default) is deprecate
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default) is deprecate
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default) is deprecate
  warnings.warn(
AdaBoost Cross-Validation Scores: [0.96666667 0.93333333 0.93333333 0.9        1.        ]
Average CV Score (AdaBoost): 0.9466666666666665
```

Mini-project on trends and applications in Data Science

Build text/ image/ video/ audio based DS Applications such as :

- **Sentiment Analysis :**

```
import nltk
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

```
[ ] # Download and load the movie reviews dataset from nltk
import nltk
nltk.download('movie_reviews')
from nltk.corpus import movie_reviews

# Load movie reviews data
documents = [(list(movie_reviews.words(fileid)), category)
             for category in movie_reviews.categories()
             for fileid in movie_reviews.fileids(category)]

# Convert to a DataFrame
df = pd.DataFrame(documents, columns=['text', 'label'])

# Join the words to create full sentences
df['text'] = df['text'].apply(lambda x: ' '.join(x))
```

```
[nltk_data] Downloading package movie_reviews to /root/nltk_data...
[nltk_data]   Unzipping corpora/movie_reviews.zip.
```


```
[nltk_data]   Unzipping corpora/movie_reviews.zip.
```


```
[ ] # Convert labels to numerical format
df['label'] = df['label'].apply(lambda x: 1 if x == 'pos' else 0)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df['text'], df['label'], test_size=0.3, random_state=42)

# Convert text to feature vectors
vectorizer = CountVectorizer()
X_train_vectorized = vectorizer.fit_transform(X_train)
X_test_vectorized = vectorizer.transform(X_test)
```

```
[ ] # Initialize and train the Naive Bayes classifier
model = MultinomialNB()
model.fit(X_train_vectorized, y_train)
```

```
MultinomialNB
MultinomialNB()
```

```
[ ] # Predict on the test set
```

```
[ ] # Predict on the test set
    y_pred = model.predict(X_test_vectorized)

    # Print the predictions
    print("Predictions:", y_pred)
```

```
Predictions: [1 1 1 1 1 1 0 1 0 1 0 0 0 0 0 0 1 0 1 1 0 0 1 1 0 1 1 0 1 0 0 0 0 0 0 0 1
 1 1 0 0 1 0 1 1 0 0 0 0 1 1 0 1 0 1 1 1 1 1 0 1 0 1 0 0 1 0 1 0 1 1 0 1 0
 0 1 1 0 0 0 0 0 1 0 1 1 1 0 0 1 1 1 0 0 0 1 0 1 1 1 0 0 0 1 1 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 1 0 0 1 0 1 0 0 0 1 0 0 1 0 1 1 1 0 0 0
 0 1 1 0 0 1 0 0 0 1 0 1 0 0 1 1 1 0 1 0 0 1 1 0 0 1 0 1 0 1 0 1 1 1 1 0 0
 0 1 1 1 1 1 0 1 0 0 0 1 0 1 1 1 0 0 0 1 1 0 1 1 0 1 0 0 0 0 1 1 0 0 1 0 0
 0 1 1 1 0 0 1 0 1 0 1 1 1 0 1 1 1 0 0 0 0 0 1 0 0 0 1 0 0 1 1 0 1 0 1 0 1
 1 0 0 1 1 1 0 0 0 0 1 1 1 0 1 0 1 0 0 1 0 1 1 0 1 1 1 0 1 0 1 1 1 1 1 0 1
 0 1 0 1 0 0 1 1 0 1 1 1 0 0 1 1 1 0 1 0 0 1 1 1 0 0 1 1 0 0 1 0 1 0 0 0 0
 1 0 1 0 0 0 1 1 0 1 0 0 1 0 0 0 1 0 0 1 1 0 0 1 1 1 0 0 0 0 1 0 0 1 0 0 0
 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 0 1 1 0 1 1 1 0 0 0 0 1 1 0 0 0 1 0 0 0 1 1
 0 0 1 0 0 0 1 0 1 0 0 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 0 1 1 0 0 0 1 0 0 1 0
 0 0 0 0 0 1 0 1 0 0 1 1 0 1 1 0 0 1 0 1 0 0 0 0 0 1 0 0 0 1 1 0 0 0 1 1 1
 1 0 0 1 1 0 0 1 1 1 0 1 0 1 0 1 0 1 1 1 0 0 0 1 1 1 0 1 1 0 0 0 1 0 1 0 0
 1 1 1 0 1 0 0 0 0 0 1 1 1 1 0 0 0 1 1 1 0 1 0 1 1 1 1 0 1 1 0 1 0 0 0 1 0
 1 1 1 0 1 1 1 0 0 0 1 1 0 1 1 1 1 0 1 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 0
 1 0 1 0 1 1 0 0]
```

```
[ ] # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)
    print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.81
```