TASK THREE: NATURAL LANGUAGE PROCESSING (NLP)

Objective: Natural Language Processing (NLP) is a subfield of Artificial Intelligence (AI) that focuses on enabling computers to understand, interpret, and generate human language. This task involves working on a specific NLP problem, such as sentiment analysis or text classification.

Code:

```
# Importing required libraries
import nltk
from nltk.corpus import movie_reviews
import random
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

# Download necessary nltk datasets
nltk.download('movie_reviews')
nltk.download('punkt')

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

# Shuffle the dataset
random.shuffle(documents)

# Feature extraction: Convert words to a single string for
each review
X = [' '.join(doc) for doc, _ in documents]
y = [category for _, category in documents]
```

```
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Convert text data into word counts using CountVectorizer
vectorizer = CountVectorizer(stop_words='english')
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

# Train a Naive Bayes classifier
clf = MultinomialNB()
clf.fit(X_train_vec, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test_vec)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
print('Classification Report:')
print(classification_report(y_test, y_pred))
```

Explanation:

- **Dataset:** The code utilizes the `movie_reviews` dataset from the `nltk` library, which contains positive and negative movie reviews.
- **Preprocessing:** It transforms each review's words into a string, where each document (review) is represented as a space-separated list of words.
- **Train/Test Split:** The dataset is divided into 80% for training and 20% for testing.
- **Vectorization:** The `CountVectorizer` converts the textual data into a matrix of token counts, essentially creating a bag-of-words representation.
- **Model:** The classification model used here is `MultinomialNB` (Naive Bayes), which is a commonly employed model for text classification tasks.

- **Evaluation:** The model's accuracy is computed, and a detailed classification report (precision, recall, F1-score) is printed.

Output: The output will display the accuracy of the model and a classification report showing precision, recall, and F1-score for both the positive and negative classes.

```
Accuracy: 82.00%
Classification Report:
              precision    recall  f1-score   support

         neg       0.78      0.87      0.82       193
         pos       0.86      0.77      0.82       207

    accuracy                           0.82       400
   macro avg       0.82      0.82      0.82       400
weighted avg       0.82      0.82      0.82       400
```

TASK SIX: REINFORCEMENT LEARNING FOR GAME PLAYING

Objective: Reinforcement Learning (RL) is a branch of AI that focuses on training agents to make sequential decisions in an environment to maximize cumulative rewards. This task involves implementing and training an RL agent to play a complex game.

Code:

```
# Import the required libraries

import numpy as np

import gym



# Initialize the environment: FrozenLake with render_mode
specified

env = gym.make('FrozenLake-v1', is_slippery=False,
render_mode='human')
```

```python
# Initialize Q-table with zeros
Q = np.zeros([env.observation_space.n, env.action_space.n])


# Set hyperparameters
learning_rate = 0.8
discount_factor = 0.95
episodes = 1000
max_steps = 100  # Max steps per episode
exploration_rate = 1.0  # Exploration rate (epsilon)
max_exploration_rate = 1.0
min_exploration_rate = 0.01
decay_rate = 0.001


# List to store total rewards per episode
rewards_all_episodes = []


# Q-learning algorithm
for episode in range(episodes):
    state = env.reset()[0]  # Reset the environment and get initial state
    total_rewards = 0
```

```
    for step in range(max_steps):
        # Choose action using epsilon-greedy approach
        if np.random.rand() < exploration_rate:
            action = env.action_space.sample()  # Explore
        else:
            action = np.argmax(Q[state, :])  # Exploit

        # Take the action and observe the new state and
reward
        new_state, reward, done, _, _ = env.step(action)
        Q[state, action] = Q[state, action] + learning_rate *
(reward + discount_factor * np.max(Q[new_state, :]) -
Q[state, action])

        state = new_state  # Update the state
        total_rewards += reward

        if done:  # If the episode is over, break the loop
            break

    # Decrease exploration rate (epsilon) to reduce
exploration over time
```

```
    exploration_rate = min_exploration_rate +
(max_exploration_rate - min_exploration_rate) *
np.exp(-decay_rate * episode)


    rewards_all_episodes.append(total_rewards)


# Evaluate the learned policy

print(f"Average reward per 100 episodes:
{np.mean(rewards_all_episodes[-100:])}")


# Test the learned policy

state = env.reset()[0]

done = False

while not done:

    action = np.argmax(Q[state, :])  # Take the best action
according to Q-table

    state, reward, done, _, _ = env.step(action)

    env.render()  # Render the environment (show the grid)
```

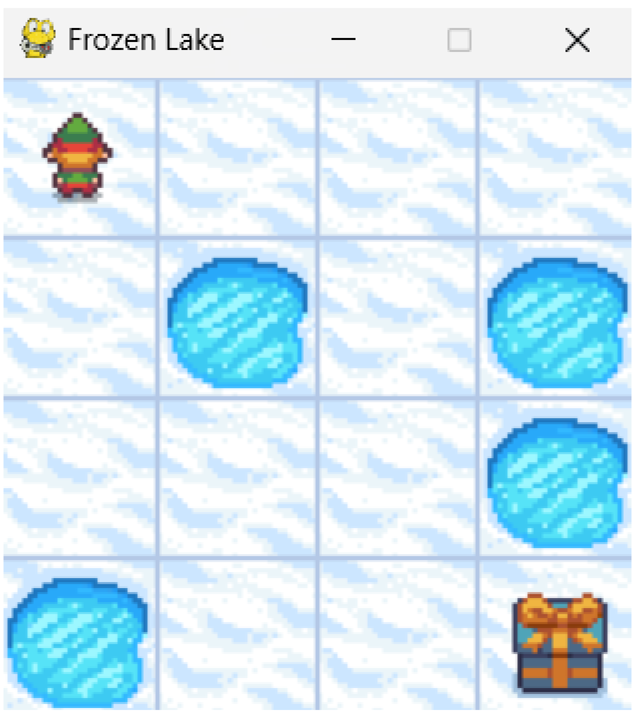
Explanation:

- **Environment Setup:** The agent plays the FrozenLake game from the OpenAI Gym library, which is a grid world where the agent needs to navigate without falling into holes.

- **Q-table Initialization:** A Q-table is initialized with zeros, where the rows represent states and the columns represent actions.
- **Hyperparameters:** The learning rate, discount factor, exploration rate, and decay rate are set to guide the learning process.
- **Q-learning Algorithm:** The agent explores the environment using an epsilon-greedy approach, updates the Q-values based on the rewards, and refines its policy over episodes.
- **Evaluation:** The average reward per 100 episodes is computed, and the learned policy is tested by rendering the environment with the best action chosen by the Q-table.

Output:

- The output will show the average reward per 100 episodes and render the environment while testing the learned policy.



Conclusion:

- **Task Three (NLP):** The sentiment analysis model uses the Naive Bayes classifier to predict whether movie reviews are positive or negative based on the content.
- **Task Six (Reinforcement Learning):** The Q-learning agent learns to navigate the FrozenLake environment and maximizes its cumulative rewards by exploring and exploiting actions.