Intellihack NextGen

Task 02

Group Name: Cryptonovate

Contents

Question 02

Imagine you are tasked with building a simple Natural Language Understanding (NLU) system to classify intents from text input. Your goal is to train a model by giving text examples for different intents and then test it with random texts, providing an intent classification along with a confidence score. You are also required to implement a fallback mechanism in case the confidence level does not meet a certain threshold.

Scenario:

You have to create a dataset containing examples of different intents for a specific domain along with their corresponding labels. For example:

• Intent: Greet

o Examples: "Hi", "How are you?", "Hello"

• Intent: Farewell

o Examples: "Goodbye", "See you later", "Take care"

• Intent: Inquiry

o Examples: "What's the weather like today?", "Can you tell me the time?", "Where is the

nearest restaurant?"

You are required to train a model using the created dataset to classify intents from new text inputs. Once trained, your model should be able to classify intents with a confidence score. If the confidence score for a classification exceeds a predefined threshold (e.g., 0.7), your system should return the predicted intent along with the confidence score. If the confidence score does not meet this threshold, your system should return a fallback response indicating that the intent could not be confidently determined.

Tasks:

1. Implement the training process for your intent classification model using a suitable machine learning or natural language processing library (e.g., scikit-learn, TensorFlow, PyTorch).

2. Develop a function to classify intents from new text inputs using the trained model. This function should return the predicted intent along with a confidence score.

3. Implement a fallback mechanism in your classification function. If the confidence score for the predicted intent is below the predefined threshold, return a fallback response (e.g., "NLU fallback: Intent could not be confidently determined").

4. Test your model and classification function with random text inputs to ensure they provide the expected intent classifications along with confidence scores or fallback responses.

Submission Requirements:

Submit your Python code implementing the training process, intent classification function, and fallback mechanism.

Include a brief explanation of your approach, detailing the choice of model, any preprocessing steps, and the rationale behind the confidence threshold selected for fallback.

Note: You may use any programming language and libraries of your choice

Code

```
import json
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
import numpy as np

import pandas as pd

with open('intent_dataset.json', 'r') as file:
    dataset = json.load(file)

examples = []
labels = []
for intent, texts in dataset.items():
    examples.extend(texts)
    labels.extend([intent] * len(texts))

vectorizer = CountVectorizer()
X = vectorizer.fit_transform(examples)
classifier = LogisticRegression(max_iter=1000)
classifier.fit(X, labels)

def classify_intent(text):
    text_vectorized = vectorizer.transform([text])
    confidence_level = classifier.predict_proba(text_vectorized)
    max_confidence = np.max(confidence_level)
    predicted_intent = classifier.predict(text_vectorized)[0]

    return predicted_intent, max_confidence

def classify_intent_with_fallback(text, threshold=0.7, fallback_response="NLU
fallback: Intent could not be confidently determined"):
    text_vectorized = vectorizer.transform([text])
    confidence_level = classifier.predict_proba(text_vectorized)
    max_confidence = np.max(confidence_level)
    predicted_intent = classifier.predict(text_vectorized)[0]

    if max_confidence >= threshold:
        return predicted_intent, max_confidence
    else:
        return fallback_response, max_confidence

phrase = input("Enter a phrase: ")
```

```
intent, confidence = classify_intent_with_fallback(phrase)
print("Text: ", phrase)
print("Intent: ", intent)
print("Confidence: ", round(confidence, 2))
```

Code Explanation

1. Importing Libraries:

- **`json`:** Used for reading JSON files.

- **`CountVectorizer`** from **`sklearn.feature_extraction.text`:** Converts text data into a matrix of token counts.

- **`LogisticRegression`** from **`sklearn.linear_model`:** Implements logistic regression for classification tasks.

- **`numpy as np`:** Used for numerical computations.

- **`pandas as pd`:** Not used in this code snippet, seems to be imported unnecessarily.

2. Loading Dataset:

- The code loads a dataset from a JSON file named **`intent_dataset.json`.** This dataset contains examples of different intents along with their corresponding labels.

3. Data Preparation:

- It prepares the dataset by extracting examples and their corresponding labels from the loaded JSON file**.**

4. Vectorization and Training:

- **`CountVectorizer`** is used to convert text examples into a matrix representation suitable for training the model.

- The text examples are vectorized using **`CountVectorizer`,** and a logistic regression classifier is trained on the vectorized data. The classifier is trained to predict intents based on the vectorized text examples.

5. Classification Functions:

- **`classify_intent(text)`:** This function takes a text input, vectorizes it using the trained **`CountVectorizer`**, predicts the intent using the trained **`LogisticRegression`** classifier, and returns the predicted intent along with the maximum confidence score.

  - `classify_intent_with_fallback(text, threshold=0.7, fallback_response="NLU fallback: Intent could not be confidently determined")`: This function extends the previous one by adding a fallback mechanism. If the confidence score for the predicted intent is below a predefined threshold, it returns a fallback response instead of the predicted intent.

6. User Input and Classification:

  - The code prompts the user to input a phrase.

  - It then classifies the intent of the input phrase using the **`classify_intent_with_fallback`** function.

  - Finally, it prints out the input text, predicted intent, and confidence score.

This code essentially creates a simple NLU system for intent classification using logistic regression, with an option to fallback to a default response if the confidence in the classification is below a certain threshold.

Output

1. Input Phrase:

  - The input phrase provided by the user is "Howdy there, what's up?". This is the text input that the model is tasked with classifying into one of the predefined intents.

2. Predicted Intent:

  - The model predicts the intent of the input phrase to be "Greet". This means that based on the text input, the model determines that the user's intention is to greet or initiate a conversation.

3. Confidence Score:

  - The confidence score associated with the predicted intent is 0.85. This score represents the model's level of certainty or confidence in its prediction. In this case, a confidence score of 0.85 indicates that the model is 85% confident in its prediction that the intent is "Greet".

4. Interpretation:

  - The output suggests that the model is quite confident in its prediction. With a confidence score of 0.85, the model is highly certain that the user's intent is to greet. This level of confidence surpasses the predefined threshold of 0.7, indicating that the model's prediction is reliable and can be trusted.

5. Response Evaluation:

- Based on the output, the model successfully interprets the user's input as a greeting. This demonstrates the effectiveness of the NLU system in accurately classifying intents from text inputs. The high confidence score further reinforces the reliability of the model's prediction.

In summary, the output provides detailed insights into how the model interprets the input text, predicts the intent, assigns a confidence score to the prediction, and determines whether the prediction meets the predefined threshold for confidence. This information helps users understand the model's decision-making process and assess the reliability of its predictions.

Model Used

The choice of the scikit-learn library for building the intent classification model in the provided code offers several advantages:

Ease of Use: Scikit-learn provides a user-friendly interface for building machine learning models, making it accessible to both beginners and experienced practitioners. The API is well-documented and intuitive, allowing for rapid prototyping and experimentation.

Rich Set of Algorithms: Scikit-learn offers a wide range of machine learning algorithms for classification, regression, clustering, dimensionality reduction, and more. For this task, logistic regression was chosen, which is a simple yet effective algorithm for binary and multiclass classification tasks.

Efficiency: Scikit-learn is implemented in Python and is built on top of other scientific libraries such as NumPy, SciPy, and Cython. It is optimized for performance and efficiency, making it suitable for processing large datasets efficiently.

Community Support: Scikit-learn has a large and active community of users and developers. This means that there are plenty of resources available online, including documentation, tutorials, and community forums where users can seek help and advice.

Integration with Other Libraries: Scikit-learn integrates well with other Python libraries commonly used in data science and machine learning workflows, such as Pandas for data manipulation, Matplotlib and Seaborn for data visualization, and TensorFlow or PyTorch for deep learning tasks. This makes it easy to incorporate scikit-learn models into more complex pipelines if needed.

Overall, scikit-learn is a versatile and reliable library that offers a solid foundation for building and deploying machine learning models, making it a suitable choice for the intent classification task described in the code.

The Rationale Behind the Confidence Threshold Selected for Fallback.

Selecting the confidence threshold of 0.7 for fallback in the intent classification system involves considering several factors:

Balancing Precision and Recall: A threshold of 0.7 ensures that the system is conservative in making predictions, preferring to only classify inputs when it's relatively confident. This helps minimize the risk of providing incorrect classifications to the user, thereby improving precision.

Maintaining High Confidence: A threshold of 0.7 ensures that only predictions with a high degree of confidence (70% or higher) are returned directly to the user. This helps maintain a high level of confidence in the responses provided by the system, enhancing user trust and satisfaction.

Reducing False Positives: By setting a relatively high threshold, the system aims to minimize the number of false positives (incorrect classifications). This is important in applications where inaccurate responses could have negative consequences or impact user experience adversely.

Handling Uncertainty: Natural language understanding systems often encounter inputs that are ambiguous or unclear. By setting a threshold of 0.7, the system acknowledges its uncertainty when faced with such inputs and opts for a fallback response rather than making potentially incorrect classifications.

User Expectations: Users typically expect accurate and reliable responses from natural language understanding systems. A confidence threshold of 0.7 reflects a commitment to meeting these expectations by ensuring that only highly confident predictions are returned to the user.