api

July 3, 2024

[ ]:
```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from joblib import load
from sklearn.metrics.pairwise import cosine_similarity
from fuzzywuzzy import fuzz

app = FastAPI()

# Load the vectorizer and classifier
vectorizer = load('qa_vectorizer.joblib')
clf = load('qa_classifier.joblib')

training_data = [
    ("What is your return policy?", "Our return policy lasts 30 days."),
    ("Tell me something about your return policy?", "Yes Sure, we would love to␣
 ↪help you out.Our return policy lasts 30 days."),
    ("How can I track my order?", "You can track your order by logging into␣
 ↪your account."),
    ("Do you offer international shipping?", "Yes, we offer international␣
 ↪shipping."),
    ("What payment methods do you accept?", "We accept credit cards (Visa,␣
 ↪MasterCard, American Express) and PayPal."),
    ("How long does shipping take?", "Shipping usually takes 3-5 business days␣
 ↪within the US."),
    ("Can I return my order if I'm not satisfied?", "Yes, we offer a 30-day␣
 ↪return policy for unused items."),
    ("Do you provide customer support?", "Yes, our customer support team is␣
 ↪available 24/7."),
    ("Are your products eco-friendly?", "Yes, all our products are made from␣
 ↪sustainable materials."),
    ("How do I change my account password?", "You can change your password in␣
 ↪the account settings."),
    ("What is your refund policy?", "We provide refunds for returned items␣
 ↪within 15 days of purchase."),
    ("Do you offer discounts for bulk orders?", "Yes, we offer discounts for␣
 ↪bulk purchases. Please contact our sales team for details."),
```

```
    ("Where are your products manufactured?", "Our products are manufactured␣
 ↪locally in the United States."),
    ("How can I contact your support team?", "You can contact our support team␣
 ↪via email at support@example.com or by phone at +1-123-456-7890."),
    ("What are the shipping costs?", "Shipping costs vary based on location and␣
 ↪order size. You can view shipping costs at checkout."),
    ("Is there a warranty on your products?", "Yes, we offer a 1-year warranty␣
 ↪on all our products."),
    ("Can I cancel my order?", "You can cancel your order within 24 hours of␣
 ↪placing it. Please contact customer support for assistance."),
    ("Do you have a loyalty program?", "Yes, we offer a loyalty program that␣
 ↪rewards repeat customers with discounts and special offers."),
    ("How can I check the status of my order?", "You can check your order␣
 ↪status by logging into your account and viewing your order history."),
    ("What are your business hours?", "Our business hours are Monday to Friday,␣
 ↪9:00 AM to 5:00 PM EST."),
    ("Do you ship internationally?", "Yes, we offer international shipping to␣
 ↪most countries. Shipping times and costs may vary."),
    ("How do I apply a coupon code?", "You can apply a coupon code at checkout␣
 ↪before completing your purchase."),
    ("What do I do if my package is lost?", "If your package is lost, please␣
 ↪contact our support team immediately for assistance."),
    ("Are there any restrictions on returns?", "Items must be in unused␣
 ↪condition with original packaging for returns. Some items may be␣
 ↪non-returnable."),

]

questions, answers = zip(*training_data)

def calculate_fuzzy_match(question, trained_question):
    return fuzz.token_set_ratio(question.lower(), trained_question.lower())

# Define a function to predict answers
def predict_answer(question):
    # Transform the question using the vectorizer
    question_vector = vectorizer.transform([question])

    # Calculate cosine similarity with trained questions
    similarity_scores = cosine_similarity(question_vector, vectorizer.
 ↪transform(questions)).flatten()

    # Calculate fuzzy match scores
    fuzzy_scores = [calculate_fuzzy_match(question, q) for q in questions]

    # Combine both scores with equal weight
```

```
    combined_scores = [(cosine + fuzz_score / 100) / 2 for cosine, fuzz_score␣
 ↪in zip(similarity_scores, fuzzy_scores)]

    # Get the index of the most similar question
    best_match_index = combined_scores.index(max(combined_scores))
    best_match_score = max(combined_scores)

    # Check if the best match score meets a certain threshold
    if best_match_score < 0.4:  # Adjust threshold as needed
        return "Out of scope"

    # Return the answer of the best matching question
    return answers[best_match_index]

class Query(BaseModel):
    question: str

@app.post("/predict")
async def predict(query: Query):
    question = query.question
    if not question:
        raise HTTPException(status_code=400, detail="No question provided")
    answer = predict_answer(question)
    return {"question": question, "answer": answer}

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```