Department of Computer Engineering

T.E. (Computer Sem VI)

Artificial Intelligence (CSC604)

Student Name: Nimish Ravindra Patil

Roll No: 9565

Assignment 2:

Considering the fallowing objectives :

CSC604.1: To grasp the fundamental concepts and methods involved in creating intelligent systems.

1. CSC604.2: Ability to choose an appropriate problem solving method and knowledge representation technique.
2. CSC604.3: Ability to analyze the strength and weaknesses of AI approaches to knowledge–intensive problem solving.
3. CSC604.4: Ability to design models for reasoning with uncertainty as well as the use of unreliable information.
4. CSC604.5: Ability to design and develop AI applications in real world scenarios.

A) what are the key considerations in designing an expert system that effectively utilizes knowledge representation techniques to handle uncertainty and unreliable information, while ensuring practicality in real-world applications?

B) Additionally, how do these considerations align with the strengths and weaknesses of various AI approaches to knowledge-intensive problem solving?"

1. Rubrics for the First Assignments:

Indicator	Average	Good	Excellent	Marks
Organization (2)	Readable with some missing points and structured (1)	Readable with improved points coverage and structured (1)	Very well written and fully structured	
Level of content(4)	All major topics are covered, the information is accurate (2)	Most major and some minor criteria are included. Information is accurate (3)	All major and minor criteria are covered and are accurate (4)	
Depth and breadth of discussion and representation(4)	Minor points/information maybe missing and representation isminimal (1)	Discussion focused on some points and covers themadequately (2)	Information is presented indepth and is accurate (4)	
Total				

Signature of the Teacher

Project Report: AI-Based Product Recommendation System

Introduction:

The aim of this project was to develop an intelligent system that provides personalized product recommendations to users based on their profiles and past queries. The system utilizes both rule-based expert system techniques and machine learning methods, specifically Singular Value Decomposition (SVD), to generate recommendations.

Project Overview:

The core of the system is the ExpertSystem class, which implements rule-based decision-making using a set of predefined rules. These rules are designed to capture various user preferences and characteristics, such as age and income level, and recommend products accordingly.

Additionally, the system incorporates SVD to analyze users' past queries and identify patterns in their search behavior. By combining recommendations from both the expert system and SVD, the system aims to provide more accurate and diverse suggestions to users.

Code

```
from django.http import JsonResponse

from django.utils.decorators import method_decorator

from django.views.decorators.csrf import csrf_exempt

from django.contrib.auth.models import User as UserModel

from rest_framework.views import APIView

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.decomposition import TruncatedSVD

import numpy as np


class ExpertSystem:
```

```
def __init__(self, rules):
    self.rules = rules

def recommend_products(self, user_profile, past_queries):
    recommendations_expert = self._recommend_by_profile(user_profile)
    recommendations_svd = self._recommend_by_svd(past_queries)
    all_recommendations = list(set(recommendations_expert + recommendations_svd))
    return all_recommendations

def _recommend_by_profile(self, user_profile):
    recommendations = []
    for rule in self.rules:
        if rule.applies(user_profile):
            recommendations.extend(rule.get_recommendations())
    return recommendations

def _recommend_by_svd(self, past_queries):
    tfidf_vectorizer = TfidfVectorizer()
    tfidf_matrix = tfidf_vectorizer.fit_transform(past_queries)
    n_components = min(tfidf_matrix.shape) - 1
    svd = TruncatedSVD(n_components=n_components, random_state=42)
    svd_matrix = svd.fit_transform(tfidf_matrix)
    query_index = len(past_queries) - 1
    similar_queries_indices = np.argsort(svd_matrix[query_index])[-3:-1]
    recommendations_svd = [past_queries[i] for i in similar_queries_indices]
```

```
        return recommendations_svd


class RecommendationRule:
    def __init__(self, condition, recommendations):
        self.condition = condition
        self.recommendations = recommendations

    def applies(self, user_profile):
        return self.condition(user_profile)

    def get_recommendations(self):
        return self.recommendations

rules = [
    RecommendationRule(
        condition=lambda profile: profile['age'] < 30 and profile['income'] > 50000,
        recommendations=['Smartphone', 'Laptop']
    ),
    RecommendationRule(
        condition=lambda profile: profile['age'] >= 30 and profile['income'] > 70000,
        recommendations=['Tablet', 'Smartwatch']
    )
]

additional_rules = [
```

```
    RecommendationRule(
        condition=lambda profile: profile['age'] < 25 and profile['income'] >= 30000 and profile['income'] <= 50000,
        recommendations=['Mid-range Smartphone', 'Gaming Console']
    ),
    RecommendationRule(
        condition=lambda profile: profile['age'] >= 25 and profile['age'] < 35 and profile['income'] > 50000 and profile['income'] <= 80000,
        recommendations=['Wireless Headphones', 'Smart Home Assistant']
    ),
    RecommendationRule(
        condition=lambda profile: profile['age'] >= 35 and profile['age'] < 50 and profile['income'] > 80000 and profile['income'] <= 100000,
        recommendations=['High-end Smartwatch', 'Robot Vacuum Cleaner']
    ),
    RecommendationRule(
        condition=lambda profile: profile['age'] >= 50 and profile['age'] < 65 and profile['income'] > 60000 and profile['income'] <= 90000,
        recommendations=['Elderly-friendly Smartphone', 'Home Blood Pressure Monitor']
    ),
    RecommendationRule(
        condition=lambda profile: profile['age'] >= 65 and profile['income'] <= 60000,
        recommendations=['Large Button Phone', 'Electric Blanket']
    ),
    RecommendationRule(
        condition=lambda profile: profile['age'] >= 25 and profile['income'] > 100000,
        recommendations=['High-end Laptop', 'Smart Home Security System']
```

```
    ),
    RecommendationRule(
        condition=lambda profile: profile['age'] >= 40 and profile['age'] < 55 and profile['income'] > 70000,
        recommendations=['Outdoor Grill', 'Gardening Tools Set']
    ),
    RecommendationRule(
        condition=lambda profile: profile['age'] >= 55 and profile['income'] > 80000,
        recommendations=['Travel Backpack', 'Binoculars']
    ),
    RecommendationRule(
        condition=lambda profile: profile['age'] >= 30 and profile['age'] < 45 and profile['income'] > 60000
and profile['income'] <= 90000,
        recommendations=['Home Exercise Equipment', 'Recipe Book Set']
    ),
    RecommendationRule(
        condition=lambda profile: profile['age'] >= 45 and profile['income'] > 70000,
        recommendations=['Luxury Spa Package', 'Wine Tasting Tour']
    )
]

all_rules = rules + additional_rules
expert_system = ExpertSystem(all_rules)

@method_decorator(csrf_exempt, name='dispatch')
class RecommendationView(APIView):
```

```
def post(self, request, *args, **kwargs):
    try:
        user_email = request.data.get('currentUserEmail')
        user = UserModel.objects.get(email=user_email)
        user_profile = {'age': user.profile.age, 'income': user.profile.income}
        past_queries = list(SearchQuery.objects.filter(user=user).values_list('query', flat=True))
        recommendations = expert_system.recommend_products(user_profile, past_queries)
        return JsonResponse({'recommendations': recommendations})
    except Exception as e:
        return JsonResponse({'error': str(e)}, status=500)
```

Assumed Input and Output:

- Input:
    - User profile data, including age and income level.
    - Past search queries of the user.
- Output:
    - List of recommended products based on user profile and past queries.

Sample Input:

python

user_profile = {'age': 35, 'income': 75000}

past_queries = ['smartphone', 'laptop', 'smartwatch']

Sample Output:

{

"recommendations": ["High-end Smartwatch", "Robot Vacuum Cleaner", "Tablet"]

}

Achievement of Course Outcomes:

- CSC604.1: Fundamental Concepts and Methods: The project demonstrates an understanding of fundamental AI concepts through the implementation of an expert system and SVD for recommendation generation.
- CSC604.2: Problem-solving Method and Knowledge Representation: By using both rule-based and machine learning techniques, the project effectively selects appropriate problem-solving methods and knowledge representation techniques.
- CSC604.3: Analysis of AI Approaches: The project analyzes the strengths and weaknesses of AI approaches by combining rule-based reasoning with machine learning to address different aspects of decision-making.
- CSC604.4: Reasoning with Uncertainty: The system handles uncertainty by integrating multiple recommendation sources and adjusting rules to accommodate uncertain or unreliable data.
- CSC604.5: Real-world AI Applications: Integrating with Django and React frameworks, the project demonstrates its applicability in real-world scenarios, providing practical value through personalized product recommendations.

Conclusion:

In conclusion, the developed AI-based product recommendation system effectively utilizes both rule-based and machine learning techniques to provide personalized recommendations. By addressing uncertainty and leveraging past user behavior, the system offers practical solutions for real-world decision-making scenarios.