이루다 정책 매칭 시스템 상세 분석

1. 정책 데이터 수집 및 구조

1.1 데이터 수집 프로세스

```python
def load_government_policies():
    try:
        # 스크립트 파일 기준 상대 경로로 지정
        import os
        current_dir = os.path.dirname(os.path.abspath(__file__))
        excel_path = os.path.join(current_dir, '정부정책_임시DB.xlsx')

        # Excel 파일에서 정책 데이터 읽기
        df = pd.read_excel(excel_path, sheet_name='중앙부처')
        policies = df.to_dict('records')

        # 지자체 데이터도 추가
        df_local = pd.read_excel(excel_path, sheet_name='지자체')
        policies.extend(df_local.to_dict('records'))

        # 민간 데이터도 추가
        df_private = pd.read_excel(excel_path, sheet_name='민간')
        policies.extend(df_private.to_dict('records'))

        return policies
    except Exception as e:
        print(f"정책 데이터 로드 실패: {e}")
        return []
```

핵심 특징:

- **3개 시트 통합**: 중앙부처, 지자체, 민간 정책을 모두 포괄
- **56+개 정책 데이터**: 발표 자료에서 언급한 수량
- **pandas 활용**: Excel → dict 형태로 변환하여 검색 최적화

1.2 정책 데이터 스키마 (추정)

```python
```

```python
# Excel에서 로드되는 정책 데이터 구조
policy_record = {
    '서비스명': '청소년복지지원',
    '기관명': '보건복지부 청소년과',
    '구분': '중앙부처',  # 또는 '지자체', '민간'
    '지원내용': '자립준비청년을 위한 주거급여 및 생활비 지원',
    '지원대상': '만 18세 이상 자립준비청년',
    '신청방법': '온라인 신청 또는 관할 주민센터 방문',
    '연락처': '02-2100-4000',
    # ... 기타 필드들
}
```

2. 정책 검색 및 필터링 시스템

2.1 다중 검색 필터 구현

python

```python
@app.route('/policies')
@login_required
def policies():
    # 정부 정책 데이터 로드
    all_policies = load_government_policies()

    # 검색 및 필터링
    search_query = request.args.get('search', '').strip()
    category_filter = request.args.get('category', '').strip()
    recommended = request.args.get('recommended', False)

    filtered_policies = all_policies.copy()
```

검색 필터 종류:

1. **키워드 검색**: `search` 파라미터
2. **카테고리 필터**: `category` 파라미터 (중앙부처/지자체/민간)
3. **추천 정책**: `recommended` 파라미터 (사용자 맞춤형)

2.2 키워드 검색 로직

python

```python
# 검색어 필터링
if search_query:
    filtered_policies = [
        policy for policy in filtered_policies
        if (search_query.lower() in str(policy.get('서비스명', '')).lower() or
            search_query.lower() in str(policy.get('기관명', '')).lower() or
            search_query.lower() in str(policy.get('지원내용', '')).lower() or
            search_query.lower() in str(policy.get('지원대상', '')).lower())
    ]
```

검색 대상 필드:

- `서비스명`: 정책 제목
- `기관명`: 담당 기관
- `지원내용`: 지원 세부사항
- `지원대상`: 대상자 조건

검색 특징:

- **대소문자 무시**: `.lower()` 사용
- **부분 일치**: `in` 연산자로 포함 관계 검색
- **안전한 문자열 변환**: `str()` 래핑으로 None 값 처리

2.3 카테고리 필터링

```python
# 카테고리 필터링
if category_filter:
    filtered_policies = [
        policy for policy in filtered_policies
        if policy.get('구분', '') == category_filter
    ]
```

카테고리 종류:

- 중앙부처: 정부 부처별 정책
- 지자체: 시/도, 시/군/구 단위 정책
- 민간: 기업, 재단, NGO 지원 프로그램

3. 핵심: 개인화 추천 알고리즘

3.1 추천 시스템 진입점

```python
# 추천 정책 필터링 (사용자 프로필 기반)
if recommended:
    filtered_policies = get_recommended_policies(filtered_policies)
```

3.2 점수 기반 추천 알고리즘 상세 분석

```python
```

```python
def get_recommended_policies(policies):
    """사용자 프로필 기반 정책 추천"""
    try:
        conn = sqlite3.connect('database/iruda.db')
        cursor = conn.cursor()
        cursor.execute('''
            SELECT housing_status, income_level, support_needs
            FROM user_profiles WHERE user_id = ?
        ''', (current_user.id,))
        profile_data = cursor.fetchone()
        conn.close()

        if not profile_data:
            return policies[:10]  # 프로필 없으면 상위 10개 반환

        support_needs = json.loads(profile_data[2]) if profile_data[2] else []

        recommended = []
        for policy in policies:
            score = 0
            policy_text = (policy.get('서비스명', '') + ' ' +
                           policy.get('지원내용', '') + ' ' +
                           policy.get('지원대상', '')).lower()

            # 지원 요구사항 매칭
            for need in support_needs:
                if need == '주거지원' and ('주거' in policy_text or '임대' in policy_text):
                    score += 3
                elif need == '경제지원' and ('생계' in policy_text or '급여' in policy_text):
                    score += 3
                elif need == '취업지원' and ('취업' in policy_text or '일자리' in policy_text):
                    score += 3
                elif need == '교육지원' and ('교육' in policy_text or '학비' in policy_text):
                    score += 3
                elif need == '심리지원' and ('상담' in policy_text or '심리' in policy_text):
                    score += 2

            # 자립준비청년 대상 정책 우선
            if '자립' in policy_text or '청소년' in policy_text:
                score += 2

            if score > 0:
                recommended.append((policy, score))

        # 점수순으로 정렬하고 상위 20개 반환
        recommended.sort(key=lambda x: x[1], reverse=True)
```

```
        return [policy for policy, score in recommended[:20]]

    except Exception as e:
        print(f"추천 정책 생성 오류: {e}")
        return policies[:10]
```

4. 추천 알고리즘 세부 분석

4.1 점수 체계 (Scoring System)

python

```
점수 매칭 규칙:

고점수 (3점):
- 주거지원: '주거', '임대' 키워드 매칭
- 경제지원: '생계', '급여' 키워드 매칭
- 취업지원: '취업', '일자리' 키워드 매칭
- 교육지원: '교육', '학비' 키워드 매칭

중간점수 (2점):
- 심리지원: '상담', '심리' 키워드 매칭
- 대상자 특화: '자립', '청소년' 키워드 매칭

최종 점수 = Σ(매칭된 지원분야 점수) + 대상자 특화 점수
```

4.2 텍스트 매칭 전략

python

```
# 정책 텍스트 통합 검색
policy_text = (
    policy.get('서비스명', '') + ' ' +
    policy.get('지원내용', '') + ' ' +
    policy.get('지원대상', '')
).lower()
```

장점:

- **포괄적 검색**: 제목, 내용, 대상을 모두 검색
- **키워드 중복 허용**: 여러 필드에서 같은 키워드가 나와도 점수 중복 적용 안함

한계:

- **단순 키워드 매칭**: 의미론적 유사성 미고려
- **동의어 처리 부족**: '주택'과 '주거', '직업'과 '취업' 등 구분 못함

- **부정문 처리 불가**: "주거지원 제외" 같은 경우 잘못 매칭

4.3 발표자료와 실제 구현의 괴리점

발표에서 언급한 수요 예측 vs 실제 매칭

```python
# 발표자료: "시뮬레이션 결과 주거 79.14% > 경제 78.98% > 심리 78.36%"
# 실제 코드: 이 통계가 추천 알고리즘에 반영되지 않음

# 실제로는 모든 지원분야가 동일한 가중치 (3점 또는 2점)
# 수요 예측 결과가 점수 체계에 반영되지 않은 상태
```

5. 실제 매칭 프로세스 시뮬레이션

5.1 사용자 프로필 예시

```python
user_profile = {
    'housing_status': '자립준비청소년',
    'income_level': '50만원 이하',
    'support_needs': ['주거지원', '경제지원', '심리지원']
}
```

5.2 정책 매칭 시뮬레이션

```python
```

```
정책 A: "청소년 자립 주거급여 지원"
- 정책 텍스트: "청소년 자립 주거급여 지원 자립준비청년 대상 주거비 월 30만원 지원"
- 매칭 점수:
  * '주거지원' 매칭: +3점 ('주거' 키워드)
  * '자립준비청년' 대상: +2점 ('자립', '청소년' 키워드)
  * 총점: 5점

정책 B: "저소득층 생계급여 지원"
- 정책 텍스트: "저소득층 생계급여 지원 기초생활수급자 대상 월 생계비 지원"
- 매칭 점수:
  * '경제지원' 매칭: +3점 ('생계', '급여' 키워드)
  * 총점: 3점

정책 C: "청년 심리상담 서비스"
- 정책 텍스트: "청년 심리상담 서비스 만 18-34세 청년 대상 무료 심리상담"
- 매칭 점수:
  * '심리지원' 매칭: +2점 ('상담', '심리' 키워드)
  * '청소년' 대상: +2점 ('청년' 키워드)
  * 총점: 4점

최종 추천 순서: 정책 A (5점) > 정책 C (4점) > 정책 B (3점)
```

6. 시스템의 강점과 한계

6.1 강점

1. **다중 필터링**: 키워드, 카테고리, 개인화 추천을 조합
2. **실시간 검색**: 클라이언트 요청 시 즉시 필터링
3. **확장 가능성**: 새로운 정책 데이터 추가 용이
4. **안전한 처리**: None 값, 빈 문자열 등 예외 상황 처리

6.2 한계점

1. **키워드 의존성**: 단순 문자열 포함 여부만 확인
2. **가중치 미조정**: 발표에서 언급한 수요 데이터 미반영
3. **정적 점수 체계**: 사용자 피드백 기반 학습 없음
4. **지역 정보 미활용**: 사용자 위치와 지자체 정책 연계 없음

6.3 발표 자료 대비 구현 수준

```
python
```

```
발표에서 주장: "맞춤형 AI 추천 시스템"
실제 구현: "규칙 기반 키워드 매칭 시스템"

발표에서 주장: "수요 예측 기반 우선순위"
실제 구현: "모든 지원분야 동일 가중치"

발표에서 주장: "56+ 정책 데이터 활용"
실제 구현: "Excel 파일 기반 정적 데이터"
```

7. Q&A 대비 핵심 포인트

Q: "추천 알고리즘의 정확도는 어느 정도인가요?"

솔직한 답변: "현재는 키워드 기반 매칭으로 정확도 측정이 어렵습니다. 사용자 피드백 데이터가 없어 정량적 평가는 어렵지만, 기본적인 관련성 필터링은 가능합니다. 향후 사용자 만족도 조사와 성공률 추적을 통해 정확도를 측정하고 개선할 계획입니다."

Q: "수요 예측 결과가 추천에 반영되나요?"

기술적 답변: "발표에서 언급한 주거 79.14% > 경제 78.98% > 심리 78.36% 수요 예측은 현재 추천 알고리즘에 직접 반영되지 않았습니다. 모든 지원분야가 동일한 가중치를 가지고 있어, 다음 업데이트에서 수요 데이터 기반 가중치 조정을 적용할 예정입니다."

Q: "실제 정부 API와 연동하나요?"

현실적 답변: "현재는 Excel 파일 기반 정적 데이터를 사용하고 있습니다. 복지로 API나 정부24 API 연동을 검토하고 있지만, 데이터 형식 표준화와 실시간 업데이트 방식에 대한 추가 연구가 필요한 상황입니다."

8. 개선 방향 제안

8.1 단기 개선 (2-3주)

python

```python
# 가중치 기반 점수 시스템 도입
DEMAND_WEIGHTS = {
    '주거지원': 3.2,  # 79.14% 기준
    '경제지원': 3.0,  # 78.98% 기준
    '심리지원': 2.8   # 78.36% 기준
}

def calculate_weighted_score(need, base_score):
    return base_score * DEMAND_WEIGHTS.get(need, 1.0)
```

8.2 중기 개선 (1-2개월)

- **벡터 기반 유사도**: Word2Vec, BERT 등으로 의미론적 매칭

- **사용자 피드백 수집**: 추천 정책에 대한 만족도 평가
- **지역 기반 필터링**: 사용자 거주지와 지자체 정책 연계

8.3 장기 개선 (3-6개월)

- **협업 필터링**: 유사한 프로필 사용자의 선택 패턴 활용
- **강화학습**: 사용자 행동 기반 추천 품질 개선
- **실시간 API 연동**: 정부 정책 데이터 자동 업데이트

결론

현재 매칭 시스템은 기본적인 키워드 기반 필터링 수준이지만, 체계적인 구조와 확장 가능성을 갖추고 있습니다. 발표에서 제시한 AI 추천보다는 단순하지만, 실용적이고 안정적인 기반을 제공하고 있다고 평가할 수 있습니다.