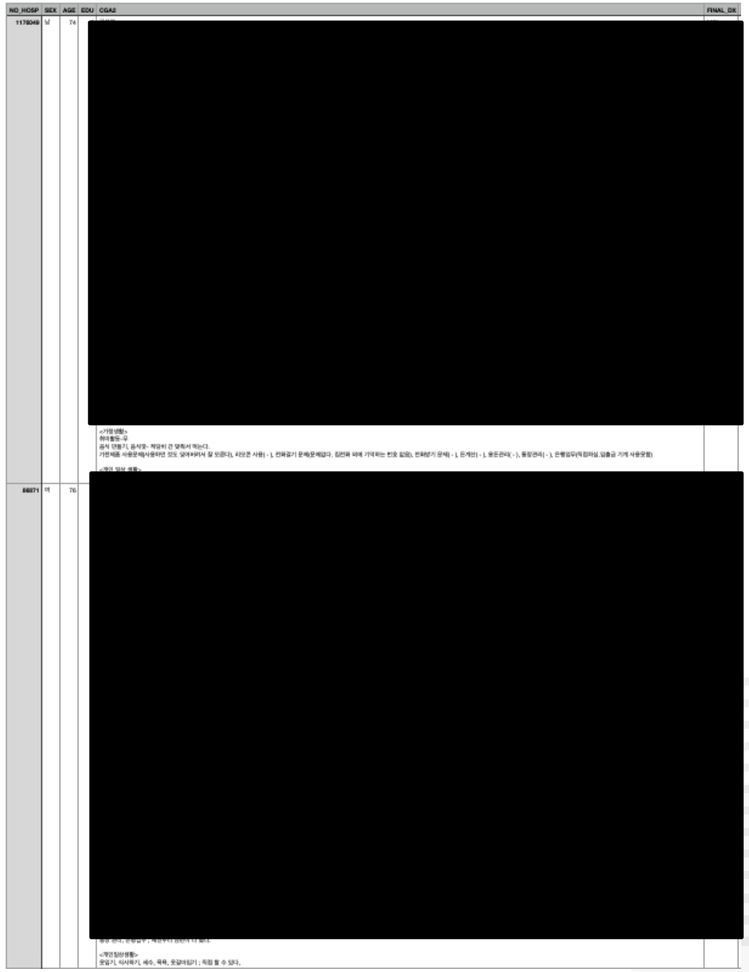
Project Mental - 2024 : PreProcessing & KoBert

Data PreProcessing

Data PreProcessing



Data PreProcessing(NC)

NO_HOSP	SEX	AGE	EDU	CQA2

Original_ID	Patient_ID	Label	Gender	Age	EDU	Education	Literacy and Numeracy	Medical History	Medication

Main Complaints	Memory	Language	Orientation	Judgment and Problem Solving	Social Activities	Home and Hobbies	Daily Living	Personality and Behavior	Mood

Family Relations

Data PreProcessing(MCI)

NO_HOSP	SEX	AGE	EDU	CQA2

Original_ID	Patient_ID	Label	Gender	Age	EDU	Education	Literacy and Numeracy	Medical History	Medication

Main Complaints	Memory	Language	Orientation	Judgment and Problem Solving	Social Activities	Home and Hobbies	Daily Living	Personality and Behavior	Additional Information

Symptoms Impact Daily Life	Basic Information

Data PreProcessing(AD)

NO_HOSP	SEX	AGE	EDU	CQA2

Original_ID	Patient_ID	Label	Gender	Age	EDU	Education	Literacy and Numeracy	Medical History	Medication

Main Complaints	Memory	Language	Orientation	Judgment and Problem Solving	Social Activities	Home and Hobbies	Daily Living	Personality and Behavior

Original_ID, EDU, Label, Sex

NO_HOSP	SEX	AGE	EDU
[redacted]			

FINAL_DX
[redacted]

```python
# CSV의 SEX, AGE, EDU로 보완
if parsed["Gender"] is None:
    sex_val = str(row.get("SEX", "")).upper()
    if sex_val in ["M","남"]:
        parsed["Gender"] = 1
    elif sex_val in ["F","여"]:
        parsed["Gender"] = 0

if not parsed["Age"]:
    age_val = row.get("AGE", None)
    if pd.notnull(age_val):
        parsed["Age"] = str(int(age_val))

if not parsed["Edu"]:
    csv_edu = row.get("EDU", None)
    if pd.notnull(csv_edu):
        parsed["Edu"] = str(int(csv_edu))
        # Education 비어있으면 생성
        if not parsed["Education"]:
            parsed["Education"] = [f"1. CSV 학력: {parsed['Edu']}년"]
```

```python
# Label (진단명 등)
label_val = row.get("FINAL_DX", None)
```

Education

학력: 중졸 (9 년)

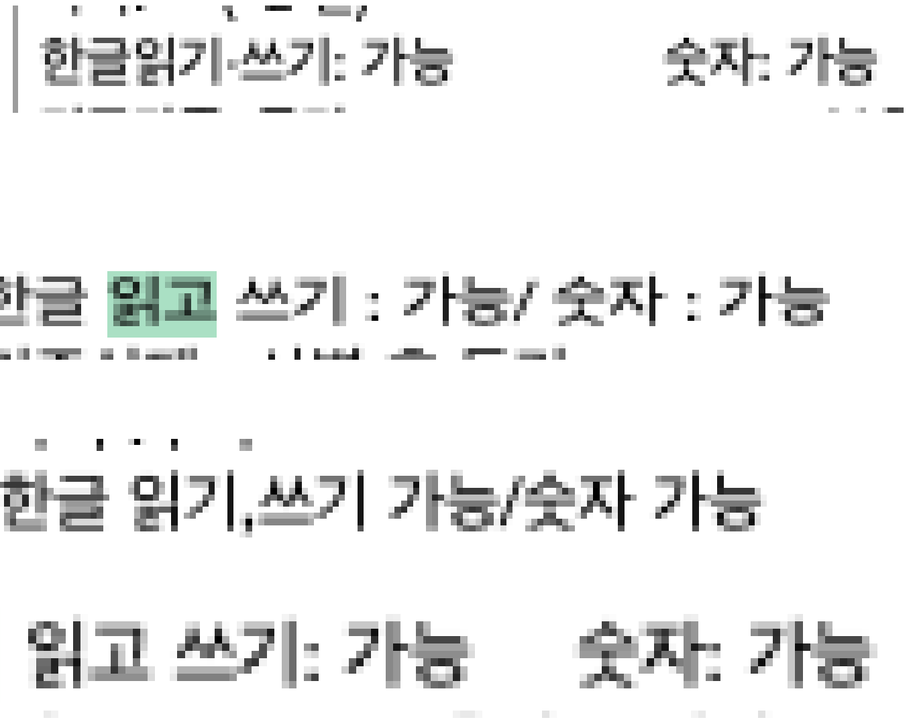
국 중퇴(3Y)

중학교 중퇴(7y)

```python
# 1) "학력:" 키워드가 있으면
if "학력" in line:
    match_edu = re.search(r"학력\s*[:]?\s*(.*)", line)
    if match_edu:
        education_line = match_edu.group(1).strip()

        # 괄호 (숫+Y), (숫+년), (숫) 등 추출
        # (A) (숫+Y)
        match_1 = re.search(r"\(\s*(\d+)\s*y\s*\)", education_line, re.IGNORECASE)
        if match_1:
            edu_year = match_1.group(1).strip()

        # (B) (숫+년)
        match_2 = re.search(r"\(\s*(\d+)\s*년\s*\)", education_line)
        if match_2:
            edu_year = match_2.group(1).strip()

        # (C) (숫+)
        match_3 = re.search(r"\(\s*(\d+)\s*\)", education_line)
        if match_3:
            edu_year = match_3.group(1).strip()
```

```python
else:
    # 2) "학력:"이 없지만 "중퇴(" 또는 "졸(" 패턴이 있는 경우
    # (A) 중퇴(숫+Y)
    match_mid = re.search(r"(.+?(중퇴|졸)\s*\(\s*(\d+)\s*y\s*\).*)", line, re.IGNORECASE)
    if match_mid:
        # 예: group(1)="국 중퇴(3Y)", group(3)="3"
        education_line = match_mid.group(1).strip()
        edu_year = match_mid.group(3).strip()
    else:
        # (B) 중퇴(숫+년) 또는 졸(숫+년)
        match_mid2 = re.search(r"(.+?(중퇴|졸)\s*\(\s*(\d+)\s*년\s*\).*)", line)
        if match_mid2:
            education_line = match_mid2.group(1).strip()
            edu_year = match_mid2.group(3).strip()
        else:
            # (C) 중퇴(숫) 또는 졸(숫)
            match_mid3 = re.search(r"(.+?(중퇴|졸)\s*\(\s*(\d+)\s*\).*)", line)
            if match_mid3:
                education_line = match_mid3.group(1).strip()
                edu_year = match_mid3.group(3).strip()
```

Literacy and Numeracy




```python
def parse_line_for_literacy(line: str):
    """
    한 줄에서 '한글 읽기·쓰기' / '읽고 쓰기' / '숫자' 등 문해력 표현을 찾아 리스트.
    예: "한글 읽고 쓰기 미숙 / 숫자 가능" -> ["한글 읽고 쓰기: 미숙", "숫자: 가능"]
    """
    items = []
    # '/'로 구분
    segments = re.split(r"/", line)
    pattern_lit = re.compile(r"(한글\s*읽기[·\s]*쓰기|읽고\s*쓰기|한글읽기·쓰기|숫자)\s*[:]?\s*(.*)")
    for seg in segments:
        seg = seg.strip()
        m = pattern_lit.search(seg)
        if m:
            key_part = m.group(1).strip()
            val_part = m.group(2).strip()
            items.append(f"{key_part}: {val_part}")
    return items
```

Main complaints

<주호소> [redacted]

호소내용 [redacted]

```python
def parse_main_complaints(text: str):
    """
    Main Complaints(주호소):
      1) <주호소> 섹션
      2) "호소내용" 키워드
    """
    main_text = ""

    # (A) <주호소> 섹션
    chunks = re.split(r"<(.*?)>", text)
    for i in range(1, len(chunks), 2):
        sec_name = chunks[i].strip()
        if sec_name == "주호소":
            main_text = chunks[i+1].strip()
            break

    # (B) "호소내용" 키워드 (콜론이 선택적)
    if not main_text:
        mc_match = re.search(r"호소내용\s*[:]?\s*(.*)", text, re.DOTALL)
        if mc_match:
            main_text = mc_match.group(1).strip()

    if main_text:
        return to_enumerated_list(main_text)
    return None
```

Medical History

<병력>
고혈압(-) 고지혈증(+) 당뇨(+)

[redacted]

<현병력>

[redacted]

50세 고혈압(+) 고지혈증(-) 당뇨(-), 20년전 협심증

[redacted]

```python
def extract_global_medical_history(text: str):
    """
    텍스트 전역에서 '그 외 특이 병력 :' 같은 표현
    또는 혈압/당뇨/고지혈증/두부 외상/뇌경색/뇌출혈 등 키워드가 있는 라인
    -> Medical History에 추가
    """
    mh_collected = []
    pattern_special = re.compile(r"그\s*외\s*특이\s*병력\s*[:]\s*(.*)", re.IGNORECASE)

    # 키워드 목록 (혈압, 당뇨, 고지혈증, 두부외상, 뇌경색, 뇌출혈 등)
    # 필요에 따라 확장
    keywords = ["혈압", "당뇨", "고지혈", "두부\s*외상", "뇌경색", "뇌출혈", "수술"]  # 예시

    lines = text.splitlines()
    for ln in lines:
        ln_str = ln.strip()
        if not ln_str:
            continue

        # (1) '그 외 특이 병력 : ...' 라인
        m_spec = pattern_special.search(ln_str)
        if m_spec:
            mh_collected.append(ln_str)
            continue

        # (2) 키워드 검색
        for kw in keywords:
            if re.search(kw, ln_str):
                # 라인 전체를 MedicalHistory로 넣기
                mh_collected.append(ln_str)
                break

    if mh_collected:
        return to_enumerated_list(mh_collected)
    return None
```

Medication


```python
def extract_global_medications(text: str):
    """
    텍스트 전역에서 <복용 중인 약물> 섹션 / '복용' / 'po중' / '약 복용' 등
    """
    # 1) <복용 중인 약물> 섹션 미리 찾기 (후에 enumerated)
    meds_collected = []

    pat_section = re.split(r"<(.*?)>", text)
    section_map = {}
    for i in range(1, len(pat_section), 2):
        raw_name = pat_section[i].strip()
        sec_content = pat_section[i+1].strip() if (i+1 < len(pat_section)) else ""
        section_map[raw_name] = sec_content

    # 만약 섹션이 존재하면
    if "복용 중인 약물" in section_map:
        raw_meds = section_map["복용 중인 약물"]
        lines_meds = raw_meds.splitlines()
        for ln in lines_meds:
            if ln.strip():
                meds_collected.append(ln.strip())

    # (B) 다른 형태 ('PO:', '복용','po중','약 복용' 등) 라인 스캔
    pat_med = re.compile(r"(PO\s*:|복용|po\s*중|약\s*복용|약물\s*복용)", re.IGNORECASE)
    lines = text.splitlines()
    prev_line_med = False
    for ln in lines:
        ln_str = ln.rstrip()
        if not ln_str:
            prev_line_med = False
            continue
        if pat_med.search(ln_str):
            meds_collected.append(ln_str)
            prev_line_med = True
        elif prev_line_med and ln.startswith(" "):  # 이전 줄이 약물 관련이면 들여쓰기 된 라인도 포함
            meds_collected.append(ln_str)
        else:
            prev_line_med = False
```

Memory, Language, Orientation, Judgment

<기억력> 0점

<언어>

```python
# (7c) <기억력> -> Memory
mem_text = sections.get("기억력", "")
if mem_text:
    result["Memory"] = to_enumerated_list(mem_text)

# (7d) <언어> -> Language
lang_text = sections.get("언어", "")
if lang_text:
    result["Language"] = to_enumerated_list(lang_text)

# (7e) <지남력> -> Orientation
ori_text = sections.get("지남력", "")
if ori_text:
    result["Orientation"] = to_enumerated_list(ori_text)

# (7f) <판단 및 문제해결> -> Judgment
judge_text = sections.get("판단 및 문제해결", "")
if judge_text:
    result["Judgment and Problem Solving"] = to_enumerated_list(judge_text)
```

Personality, Social Activities, Home and Hobbies

<성격, 행동> GDS=1

<사회/취미생활>

<가정 및 취미활동>

```python
# (7g) <성격변화>, <성격,행동>, <성격, 행동>, <성격> -> Personality
behave_text = sections.get("성격변화", "")
if not behave_text:
    behave_text = sections.get("성격,행동", "")
if not behave_text:
    behave_text = sections.get("성격, 행동", "")
if not behave_text:
    behave_text = sections.get("성격", "")
if behave_text:
    result["Personality and Behavior"] = to_enumerated_list(behave_text)

# (7h) <사회활동> -> Social Activities
social_txt = sections.get("사회활동", "")
if social_txt:
    result["Social Activities"] = to_enumerated_list(social_txt)

# (7i) <사회/취미생활> -> Social Activities + Home and Hobbies
sc_hobby_txt = sections.get("사회/취미생활", "")
if sc_hobby_txt:
    sc_enum = to_enumerated_list(sc_hobby_txt)
    result["Social Activities"] = sc_enum
    result["Home and Hobbies"] = sc_enum

# (7j) <가정생활>, <가정 및 취미활동> -> Home and Hobbies
home_txt = sections.get("가정생활", "")
if not home_txt:
    home_txt = sections.get("가정 및 취미활동", "")
if home_txt:
    result["Home and Hobbies"] = to_enumerated_list(home_txt)
```

Data PreProcessing

<가정 및 취미활동>0 횟수와 시간의 변화

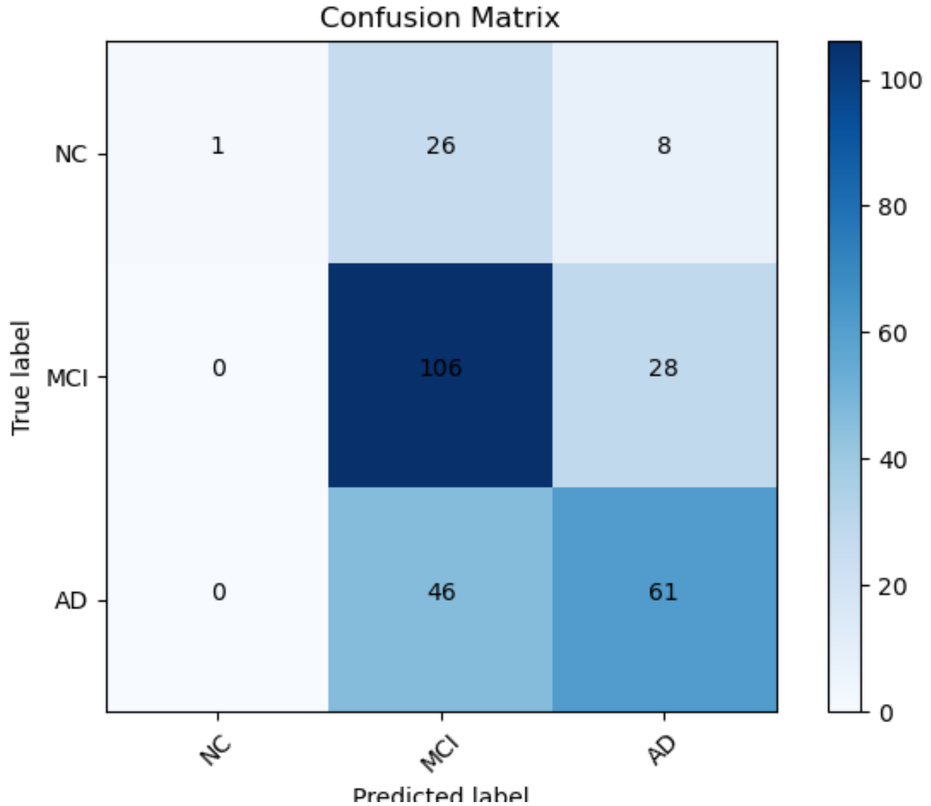
<개인일상생활> 0 스스로 완벽/조언이나 지시 필요/신체적 도움 필요

<가정생활> 0

```python
# 가정 및 취미 생활 (또는 가정생활)
combined_hobby_text = ""
# "가정 및 취미활동" 섹션이 있으면 추가
if "가정 및 취미활동" in sections:
    combined_hobby_text += sections.get("가정 및 취미활동", "")
# "가정생활" 섹션이 있으면 추가 (이미 내용이 있다면 줄바꿈 추가)
if "가정생활" in sections:
    if combined_hobby_text:
        combined_hobby_text += "\n" + sections.get("가정생활", "")
    else:
        combined_hobby_text = sections.get("가정생활", "")

if combined_hobby_text:
    lines_ = [ln.strip() for ln in combined_hobby_text.splitlines() if ln.strip()]
    if lines_:
        enumerated = []
        for i, line_ in enumerate(lines_, start=1):
            enumerated.append(f"{i}. {line_}")
        result["가정 및 취미 생활"] = enumerated


# 일상 생활 수행 능력
adl_text = sections.get("개인일상생활", "")
if adl_text:
    lines_ = [ln.strip() for ln in adl_text.splitlines() if ln.strip()]
    if lines_:
        enumerated = []
        for i, line_ in enumerate(lines_, start=1):
            enumerated.append(f"{i}. {line_}")
        result["일상 생활 수행 능력"] = enumerated
```

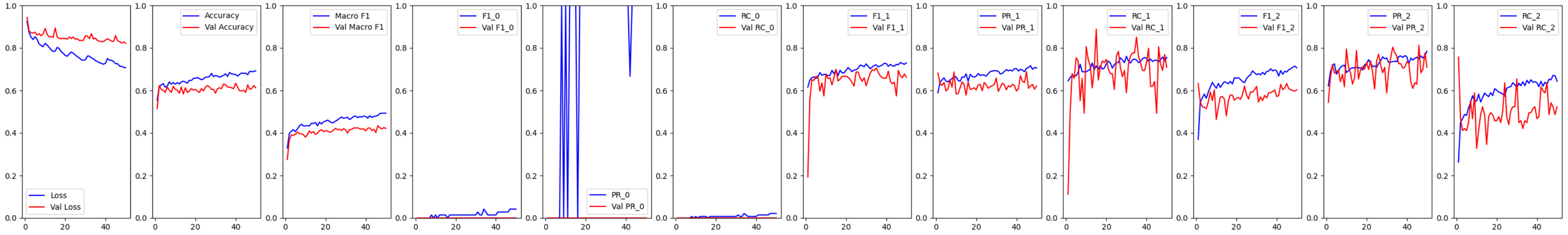
KoBert

Lexicon Feature + Hallow feature

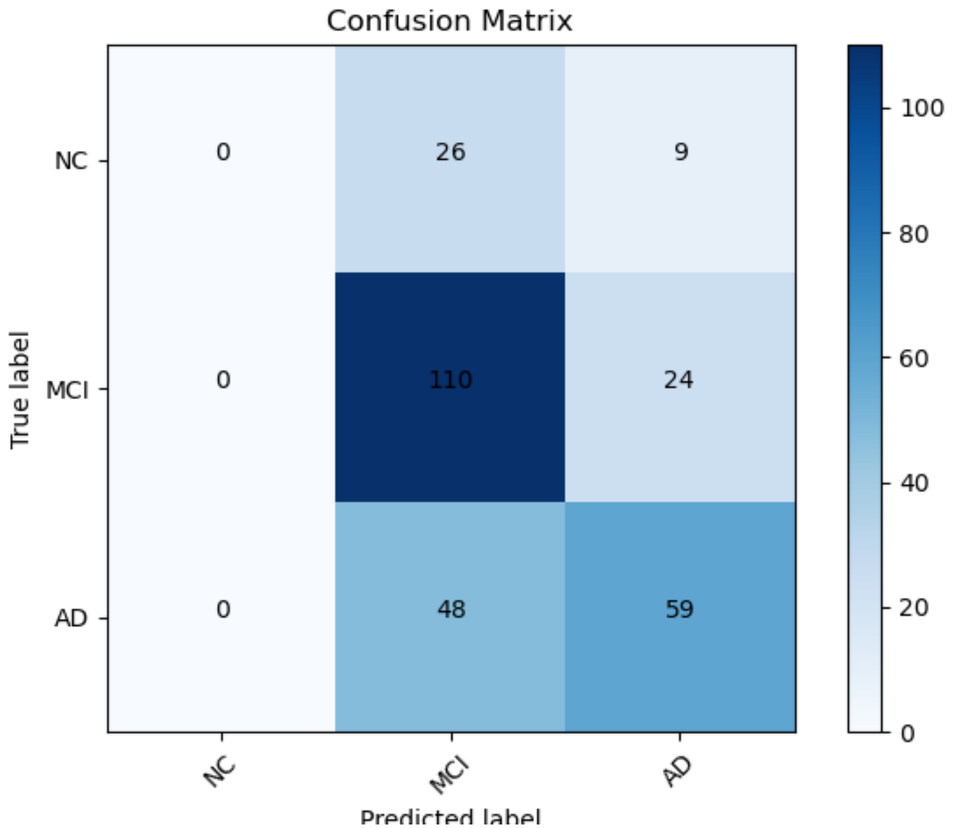



```
9/9 ━━━━━━━━━━━━━━━━━━━━ 30s 3s/step
              precision    recall  f1-score   support

           0       1.00      0.03      0.06        35
           1       0.60      0.79      0.68       134
           2       0.63      0.57      0.60       107

    accuracy                           0.61       276
   macro avg       0.74      0.46      0.44       276
weighted avg       0.66      0.61      0.57       276

Confusion matrix, without normalization
[[  1  26   8]
 [  0 106  28]
 [  0  46  61]]
```

Lexicon Feature + Hallow feature



Hallow feature



	precision	recall	f1-score	support
0	0.00	0.00	0.00	35
1	0.60	0.82	0.69	134
2	0.64	0.55	0.59	107
accuracy			0.61	276
macro avg	0.41	0.46	0.43	276
weighted avg	0.54	0.61	0.57	276

Confusion matrix, without normalization

```
[[  0  26   9]
 [  0 110  24]
 [  0  48  59]]
```

Hallow feature