

MAP3K5 IC50 활성값 예측 모델 훈련 가이드

개요

이 디렉토리는 *****약물이 얼마나 효과적인지 예측하는 AI 모델*****을 만들기 위한 도구들을 담고 있습니다.

간단히 말하면...

- **입력:** 화합물의 구조 정보 (분자식 같은 것)
- **출력:** 그 화합물이 얼마나 효과적인지 점수
- **목표:** 새로운 약물 후보물질의 효과를 미리 예측하기

데이터는 어디서 왔나요?

- **ChEMBL:** 유명한 약물 데이터베이스 (812개 화합물)
- **CAS:** 화학 논문들의 데이터 (2442개 화합물)
- **PubChem:** 공개 약물 정보 (664개 화합물)
- **총합:** 3568개의 화합물 데이터

무엇을 예측하나요?

MAP3K5라는 단백질에 대한 화합물의 **효과 강도**를 예측합니다.

- 이 단백질은 알츠하이머병, 파킨슨병 등과 관련이 있어서
- 이 단백질을 억제하는 약물을 찾는 것이 중요합니다
- **IC50**은 "얼마나 적은 양으로 효과를 보이는지"를 나타내는 지표입니다

목표

우리가 만들려는 것

"화합물 구조를 보면 약물 효과를 예측하는 AI"

입력 (AI에게 주는 정보)

- **화합물 구조:** 분자식 같은 화학 구조 정보

- 형식: SMILES (화학 구조를 문자로 표현한 것)

출력 (AI가 예측하는 것)

- 효과 점수: pIC50 값 (높을수록 더 효과적)
- 의미: "얼마나 적은 양으로 효과를 보이는지"

성능 평가 방법

- **RMSE**: 예측값과 실제값의 차이 (낮을수록 좋음)
- **R²**: 예측 정확도 (1에 가까울수록 좋음)
- **MAE**: 평균 오차 (낮을수록 좋음)

데이터 현황

- 학습 데이터: 3568개 화합물 (효과가 알려진 것들)
- 테스트 데이터: 127개 화합물 (효과를 예측할 것들)

파일 구조

```
src/train/
├── __init__.py           # 모듈 초기화 파일
├── data_preparation.py   # 데이터 준비 및 전처리 모듈
├── README.md            # 이 파일 (상세 가이드)
└── [향후 추가될 파일들]
    ├── molecular_features.py # 분자 특성 추출
    ├── model_architectures.py # 모델 아키텍처 정의
    ├── training_pipeline.py  # 훈련 파이프라인
    └── evaluation.py         # 모델 평가
```

데이터 현황 및 의미

데이터 소스 상세 분석

1. ChEMBL 데이터베이스

- 원본 데이터: 824개 화합물
- 최종 사용: 462개 화합물 (중복 제거 후)

- **특징:**
 - 체계적인 생화학적 데이터
 - 표준화된 IC50 측정값 (nM 단위)
 - 상세한 실험 조건 정보 포함
- **활성 범위:** pIC50 3.30 ~ 10.00
- **데이터 품질:** 높음 (검증된 실험실 데이터)

2. CAS (Chemical Abstracts Service) 데이터

- **원본 데이터:** 3452개 화합물
- **최종 사용:** 2442개 화합물
- **특징:**
 - 학술 논문 기반 데이터
 - 다양한 실험 조건과 측정 방법
 - 높은 활성 화합물 포함
- **활성 범위:** pIC50 6.30 ~ 16.00
- **데이터 품질:** 중간~높음 (논문 검증 데이터)

3. PubChem 데이터베이스

- **원본 데이터:** 23,795개 화합물
- **최종 사용:** 664개 화합물 (IC50 타입 필터링 후)
- **특징:**
 - 공개 생물학적 활성 데이터
 - 다양한 연구 기관의 데이터 통합
 - 대규모 스크리닝 결과
- **활성 범위:** pIC50 3.30 ~ 10.00
- **데이터 품질:** 중간 (공개 데이터베이스)



통합 데이터 통계

- **총 훈련 데이터:** 3568개 화합물
- **테스트 데이터:** 127개 화합물
- **전체 데이터셋:** 3695개 화합물
- **pIC50 범위:** 3.30 ~ 16.00 (매우 넓은 활성 범위)
- **pIC50 평균:** 9.03
- **SMILES 길이:** 21 ~ 150 문자 (평균: 65.0)

데이터 컬럼 상세 설명

핵심 컬럼

- **SMILES**: 화학 구조를 문자로 표현한 것
 - 분자식을 컴퓨터가 이해할 수 있는 형태로 바꾼 것
 - 원자들이 어떻게 연결되어 있는지 나타냄
 - 예: CC1=CC(=C(C=C1S(=O)(=O)N)C(=O)NC2=CC=CC(=N2)C3=NN=CN3C(C)C)OC
- **IC50_nM**: 효과를 나타내는 숫자
 - "얼마나 적은 양으로 효과를 보이는지"를 나타냄
 - 숫자가 작을수록 더 효과적 (적은 양으로도 효과)
 - 단위: 나노몰 (nM) - 매우 작은 단위
- **pIC50**: 효과 점수 (AI가 예측할 값)
 - IC50을 변환한 점수 (높을수록 더 효과적)
 - AI 모델이 학습하기 좋은 형태로 만든 것
 - 예: pIC50이 8.0이면 매우 효과적, 4.0이면 효과 부족

메타데이터 컬럼

- **source**: 데이터 출처 식별자
 - 'ChEMBL': ChEMBL 데이터베이스
 - 'CAS': Chemical Abstracts Service
 - 'PubChem': PubChem 데이터베이스
- **ID**: 테스트 데이터의 고유 식별자
 - 'TEST_000' ~ 'TEST_126' 형식
 - 예측 결과 매칭용

데이터셋의 의미

MAP3K5 단백질이란?

- **역할**: 우리 몸에서 세포의 생사와 염증을 조절하는 단백질
- **문제**: 이 단백질이 너무 활성화되면 질병이 생길 수 있음
- **해결책**: 이 단백질을 억제하는 약물을 찾아야 함

관련 질병들

- **알츠하이머병**: 기억력 저하, 치매
- **파킨슨병**: 떨림, 움직임 장애
- **당뇨병**: 혈당 조절 장애

- 심혈관 질환: 심장, 혈관 질환

효과 점수 (pIC50)의 의미

- 매우 효과적 (pIC50 > 8.0):
 - 적은 양으로도 큰 효과
 - 약물 개발 후보물질
- 보통 효과적 (pIC50 6.0-8.0):
 - 적당한 효과
 - 더 개선할 여지가 있음
- 효과 부족 (pIC50 < 6.0):
 - 효과가 약함
 - 구조를 바꿔서 개선 필요

사용 방법

1. 데이터 준비

```
from src.train.data_preparation import MAP3K5DataPreparation

# 데이터 준비 클래스 초기화
data_prep = MAP3K5DataPreparation(data_dir='data', random_state=42)

# 모든 데이터 로드
all_data = data_prep.load_all_data()

# 데이터 품질 검증
for data_name, data in all_data.items():
    if data is not None:
        validation_results = data_prep.validate_data_quality(data, data_name)

# 모델 훈련용 데이터셋 준비
model_datasets = data_prep.prepare_model_datasets(
    test_size=0.2,          # 검증 세트 비율
    validation_size=0.2    # 검증 세트 비율
)

# 데이터 저장
data_prep.save_prepared_data(output_dir="prepared_data")
```

2. 데이터셋 분할 결과

반환되는 데이터셋 구조

```
datasets = {  
    'train': train_df,          # 훈련 세트 (369개 화합물)  
    'validation': validation_df, # 검증 세트 (93개 화합물)  
    'test': test_df,           # 테스트 세트 (127개 화합물)  
    'full_training': full_df    # 전체 훈련 데이터 (462개 화합물)  
}
```

데이터 품질 검증

검증 항목

1. **결측치 검사**: 모든 필수 컬럼의 결측치 확인
2. **중복 검사**: SMILES 기반 중복 화합물 제거
3. **SMILES 유효성**: 길이, 패턴, 구조 검증
4. **IC50 값 범위**: 이상치 탐지 및 처리
5. **데이터 분포**: pIC50 값의 분포 분석

검증 결과 예시

 training 데이터 품질 검증 결과:

- 총 화합물 수: 462
- 중복 화합물: 0
- 결측치: {'SMILES': 0, 'IC50_nM': 0, 'pIC50': 0, 'source': 0}
- SMILES 길이: 21~106 (평균: 51.4)
- pIC50 범위: 3.30~10.00 (평균: 6.65)

데이터 전처리 파이프라인 상세

전처리 개요

데이터 전처리는 원시 데이터를 머신러닝 모델이 학습할 수 있는 형태로 변환하는 과정입니다. 각 단계별로 상세한 처리 과정을 설명합니다.

1단계: 데이터 로드 및 검증

1.1 다중 소스 데이터 로드

```
# 각 데이터 소스별 로딩 전략
def load_all_data():
    # ChEMBL: 표준화된 CSV 형식
    chembl_data = load_chembl_data()

    # CAS: 복잡한 Excel 시트 구조
    cas_data = load_cas_data_using_read_excel()

    # PubChem: 대용량 CSV, IC50 타입 필터링
    pubchem_data = load_pubchem_data_with_filtering()
```

1.2 데이터 품질 초기 검증

- **파일 존재성 확인:** 각 데이터 파일의 경로 검증
- **컬럼 구조 분석:** 필수 컬럼 존재 여부 확인
- **데이터 타입 검증:** 예상 데이터 타입과 일치 여부
- **기본 통계 분석:** 데이터 크기, 결측치 비율 등

2단계: 데이터 정제 및 표준화

2.1 SMILES 정규화

```
def normalize_smiles(smiles_series):  
    """  
    SMILES 문자열 정규화 과정  
    """  
    # 1. 공백 제거  
    normalized = smiles_series.str.strip()  
  
    # 2. 대소문자 표준화  
    normalized = normalized.str.upper()  
  
    # 3. 특수 문자 정리  
    normalized = normalized.str.replace('\\n', '')  
    normalized = normalized.str.replace('\\r', '')  
  
    # 4. 유효성 검사 (RDKit 사용)  
    valid_smiles = []  
    for smiles in normalized:  
        mol = Chem.MolFromSmiles(smiles)  
        if mol is not None:  
            valid_smiles.append(smiles)  
  
    return valid_smiles
```


2.2 IC50 값 표준화

```
def standardize_ic50_values(df):  
    """  
    IC50 값을 nM 단위로 표준화  
    """  
  
    # 1. 단위 변환  
    #  $\mu\text{M} \rightarrow \text{nM}$ :  $\times 1000$   
    #  $\mu\text{M} \rightarrow \text{nM}$ :  $\times 1$   
    #  $\text{pM} \rightarrow \text{nM}$ :  $\div 1000$   
  
    # 2. pIC50 계산  
    #  $\text{pIC50} = -\log_{10}(\text{IC50}_{\text{nM}} \times 10^{-9})$   
    df['pIC50'] = -np.log10(df['IC50_nM'] * 1e-9)  
  
    # 3. 이상치 제거  
    #  $\text{pIC50} < 3.0$  또는  $\text{pIC50} > 16.0$  제거  
    df = df[(df['pIC50'] >= 3.0) & (df['pIC50'] <= 16.0)]  
  
    return df
```

2.3 중복 제거 및 통합

```
def remove_duplicates_and_merge(datasets):  
    """  
    다중 소스 데이터 중복 제거 및 통합  
    """  
    # 1. SMILES 기반 중복 식별  
    all_smiles = []  
    for source, data in datasets.items():  
        data['source'] = source  
        all_smiles.extend(data['SMILES'].tolist())  
  
    # 2. 중복 SMILES 찾기  
    duplicates = find_duplicate_smiles(all_smiles)  
  
    # 3. 중복 제거 (가장 높은 pIC50 값 유지)  
    merged_data = []  
    for smiles in set(all_smiles):  
        # 해당 SMILES의 모든 데이터 찾기  
        all_entries = find_all_entries_for_smiles(smiles, datasets)  
  
        # 가장 높은 pIC50 값 선택  
        best_entry = max(all_entries, key=lambda x: x['pIC50'])  
        merged_data.append(best_entry)  
  
    return pd.DataFrame(merged_data)
```

3단계: 데이터 품질 검증

3.1 결측치 분석 및 처리

```
def analyze_missing_values(df):  
    """  
    결측치 상세 분석  
    """  
    missing_info = {  
        'total_rows': len(df),  
        'missing_by_column': df.isnull().sum().to_dict(),  
        'missing_percentage': (df.isnull().sum() / len(df) * 100).to_dict(),  
        'rows_with_any_missing': df.isnull().any(axis=1).sum()  
    }  
  
    # 결측치가 있는 행 제거  
    df_clean = df.dropna()  
  
    return df_clean, missing_info
```

3.2 데이터 분포 분석

```
def analyze_data_distribution(df):  
    """  
    pIC50 값 분포 분석  
    """  
    distribution_info = {  
        'pIC50_stats': {  
            'min': df['pIC50'].min(),  
            'max': df['pIC50'].max(),  
            'mean': df['pIC50'].mean(),  
            'median': df['pIC50'].median(),  
            'std': df['pIC50'].std(),  
            'skewness': df['pIC50'].skew(),  
            'kurtosis': df['pIC50'].kurtosis()  
        },  
        'source_distribution': df['source'].value_counts().to_dict(),  
        'smiles_length_stats': {  
            'min': df['SMILES'].str.len().min(),  
            'max': df['SMILES'].str.len().max(),  
            'mean': df['SMILES'].str.len().mean()  
        }  
    }  
  
    return distribution_info
```

4단계: 데이터셋 분할

4.1 스트라티파이드 샘플링

```
def stratified_split_by_pic50(df, test_size=0.2, validation_size=0.2):
    """
    pIC50 값 기반 스트라티파이드 분할
    """
    # 1. pIC50 구간별 분류
    df['pIC50_bin'] = pd.cut(df['pIC50'],
                              bins=[0, 6, 8, 10, 12, 20],
                              labels=['low', 'medium', 'high', 'very_high', 'extreme'])

    # 2. 각 구간별로 비율 유지하며 분할
    train_data, temp_data = train_test_split(
        df,
        test_size=test_size + validation_size,
        stratify=df['pIC50_bin'],
        random_state=42
    )

    # 3. 임시 데이터를 검증/테스트로 분할
    val_ratio = validation_size / (test_size + validation_size)
    validation_data, test_data = train_test_split(
        temp_data,
        test_size=1 - val_ratio,
        stratify=temp_data['pIC50_bin'],
        random_state=42
    )

    return train_data, validation_data, test_data
```

5단계: 데이터 저장 및 메타데이터 생성

5.1 표준화된 저장 형식

```
def save_prepared_data(datasets, output_dir):  
    """  
    전처리된 데이터 저장  
    """  
    # 1. CSV 파일 저장  
    for name, data in datasets.items():  
        filepath = os.path.join(output_dir, f'{name}_data.csv')  
        data.to_csv(filepath, index=False)  
  
    # 2. 데이터 요약 정보 생성  
    summary = generate_data_summary(datasets)  
    with open(os.path.join(output_dir, 'data_summary.txt'), 'w') as f:  
        json.dump(summary, f, indent=2)  
  
    # 3. 메타데이터 저장  
    metadata = {  
        'preprocessing_date': datetime.now().isoformat(),  
        'data_sources': list(datasets.keys()),  
        'total_compounds': sum(len(data) for data in datasets.values()),  
        'preprocessing_steps': [  
            'SMILES normalization',  
            'IC50 standardization',  
            'Duplicate removal',  
            'Missing value handling',  
            'Stratified splitting'  
        ]  
    }  
  
    with open(os.path.join(output_dir, 'metadata.json'), 'w') as f:  
        json.dump(metadata, f, indent=2)
```

전처리 결과 검증

검증 항목

1. 데이터 **완전성**: 모든 필수 컬럼 존재
2. 데이터 **일관성**: 단위 및 형식 통일
3. 데이터 **품질**: 이상치 및 오류 제거

4. **분포 균형**: 각 소스별 데이터 분포 확인

5. **재현성**: 랜덤 시드 설정으로 재현 가능

성공 지표

- **결측치**: 0%
- **중복**: 완전 제거
- **SMILES 유효성**: 100%
- **pIC50 범위**: 3.30 ~ 16.00
- **소스별 분포**: 균형잡힌 분포

전처리의 중요성

1. 데이터 품질의 모델 성능 영향

- **정확한 SMILES**: 분자 구조 인식 정확도 향상
- **표준화된 IC50**: 일관된 활성 예측
- **중복 제거**: 과적합 방지 및 일반화 성능 향상
- **이상치 제거**: 모델 안정성 및 예측 정확도 개선

2. 다중 소스 데이터 통합의 장점

- **데이터 다양성**: 다양한 실험 조건과 측정 방법 반영
- **활성 범위 확장**: 넓은 pIC50 범위 (3.30 ~ 16.00) 커버
- **모델 견고성**: 다양한 데이터 소스에 대한 일반화 능력
- **통계적 신뢰성**: 대규모 데이터셋으로 인한 신뢰도 향상

3. 전처리 단계별 성능 개선

원시 데이터 → 전처리 → 모델 성능

↓		↓		↓
3568개	→	정제	→	RMSE ↓
(혼재)	→	표준화	→	R ² ↑
(중복)	→	통합	→	일반화 ↑

4. 전처리 검증 체크리스트

- ☒ 모든 데이터 소스 로드 성공
- ☒ SMILES 문자열 유효성 검증
- ☒ IC50 값 단위 표준화

- ✓ 중복 화합물 제거
- ✓ 결측치 처리
- ✓ 이상치 탐지 및 제거
- ✓ 데이터 분포 균형 확인
- ✓ 스트라티파이드 분할 검증
- ✓ 메타데이터 및 요약 정보 생성



모델 개발 로드맵

Phase 1: 기본 모델 (완료 ✓)

- ✓ 데이터 로더 구현
- ✓ 데이터 품질 검증
- ✓ 데이터셋 분할
- ✓ 기본 전처리 파이프라인
- ✓ CAS 데이터 통합 (2442개 화합물 추가)
- ✓ PubChem 데이터 통합 (664개 화합물 추가)
- ✓ 다중 소스 데이터 결합 (총 3568개 화합물)

Phase 2: 분자 특성 추출 (진행 예정)

- ☐ Morgan Fingerprint 생성
- ☐ MACCS Keys 추출
- ☐ 분자 설명자 계산
- ☐ Mol2Vec 임베딩

Phase 3: 모델 아키텍처 (진행 예정)

- ☐ 전통적 ML 모델 (Random Forest, XGBoost)
- ☐ 딥러닝 모델 (MLP, GNN)
- ☐ 앙상블 모델
- ☐ 하이퍼파라미터 최적화

Phase 4: 모델 평가 (진행 예정)

- ☐ 교차 검증
- ☐ 성능 지표 계산
- ☐ 모델 해석

☐ 예측 결과 분석

환경 설정

필수 패키지

```
pip install pandas numpy scikit-learn rdkit-pypi
```

권장 패키지

```
pip install torch torch-geometric dgl  
pip install xgboost lightgbm  
pip install matplotlib seaborn plotly
```

로깅 및 모니터링

로그 레벨

- **INFO**: 일반적인 진행 상황
- **WARNING**: 주의가 필요한 상황
- **ERROR**: 오류 상황
- **DEBUG**: 디버깅 정보

로그 예시

```
2025-07-19 16:13:03,690 - INFO - MAP3K5 데이터 준비 프로세스 시작  
2025-07-19 16:13:03,706 - INFO -  테스트 데이터 로드 완료: 127개 화합물  
2025-07-19 16:13:04,497 - INFO -  훈련 데이터 로드 완료: 462개 화합물
```

성능 최적화 팁

데이터 처리 최적화

1. 캐싱 활용: `use_cache=True` 옵션으로 중복 로드 방지
2. 배치 처리: 대용량 데이터의 경우 배치 단위 처리

3. **메모리 효율성**: 필요한 컬럼만 선택하여 메모리 사용량 최소화

모델 훈련 최적화

1. **조기 종료**: 과적합 방지를 위한 early stopping
2. **학습률 스케줄링**: 적응적 학습률 조정
3. **정규화**: Dropout, BatchNorm 등 정규화 기법 활용

문제 해결

일반적인 문제들

1. 데이터 로드 실패

```
# 해결 방법: 파일 경로 확인
import os
print(os.path.exists('data/test.csv'))
print(os.path.exists('data/ChEMBL_ASK1(IC50).csv'))
```

2. 메모리 부족

```
# 해결 방법: 청크 단위 처리
def process_in_chunks(df, chunk_size=1000):
    for i in range(0, len(df), chunk_size):
        chunk = df[i:i+chunk_size]
        # 청크 처리 로직
```

3. SMILES 유효성 오류

```
# 해결 방법: RDKit을 사용한 유효성 검사
from rdkit import Chem

def validate_smiles(smiles):
    mol = Chem.MolFromSmiles(smiles)
    return mol is not None
```

참고 자료

논문 및 기술 문서

- [ChEMBL 데이터베이스](#)
- [RDKit 문서](#)
- [분자 머신러닝 가이드](#)

관련 프로젝트

- [DeepChem](#)
- [MoleculeNet](#)
- [Chemprop](#)

기여 가이드

코드 스타일

- PEP 8 준수
- 타입 힌트 사용
- 문서화 주석 작성
- 단위 테스트 작성

이슈 리포트

- 버그 리포트 시 상세한 오류 메시지 포함
- 환경 정보 (OS, Python 버전, 패키지 버전) 제공
- 재현 가능한 최소 예제 코드 제공

문의 및 지원

프로젝트 관련 문의사항이나 버그 리포트는 다음을 통해 연락해주세요:

- 이슈 트래커: [GitHub Issues](#)
- 이메일: [\[프로젝트 관리자 이메일\]](#)
- 문서: [이 README 파일 참조](#)

마지막 업데이트: 2025-07-19

버전: 1.0.0

작성자: AI Drug Discovery Team