PharmApp Suite

🧠 AI for Drug Discovery and Development 🍼

🧩 Topic: Comparing ChEMBL scaffolds with FDA drugs

Okay, I understand. You're working on a Chembl 35 data analysis project using RDKit within an AIMLops structure. You're encountering some issues with your SQL query and scikit-learn version, and you want guidance on structuring your code and running the analysis. I will provide SQL and Python code snippets, addressing your specific concerns and incorporating the AIMLops structure.

Here's a breakdown and guidance, first in English, then in Vietnamese:

English Analysis & Guidance:

1. Analysis Model & Workflow:

The overall workflow involves the following steps:

- **Data Extraction (SQL):** Extract relevant data from the Chembl 35 database using SQL queries. This will involve filtering the data based on activity values, target information, and other relevant criteria. You'll save this data as CSV files in your `data` directory.
- **Data Loading & Preprocessing (Python):** Load the CSV data into pandas DataFrames. Clean and preprocess the data, handling missing values, converting data types, and potentially applying transformations.
- **Feature Engineering (Python/RDKit):** Generate relevant molecular features using RDKit. This could include properties like molecular weight, LogP, number of hydrogen bond donors/acceptors, topological polar surface area (TPSA), and various fingerprints (e.g., Morgan fingerprints).
- **Data Analysis and Modeling (Python):** Perform exploratory data analysis (EDA) to understand the relationships between features and activity. Build predictive models (e.g., linear regression, random forest, etc.) to predict activity based on molecular features.
- **Evaluation:** Evaluate the performance of the models using appropriate metrics (e.g., RMSE, R-squared).

2. Code Structure and AIMLops Integration

You are using an AIMLOps template. Great! This usually has a structure like:

```
project_root/
├── data/
│   ├── raw/          # Raw data from SQL (CSV files)
│   ├── processed/    # Processed data ready for modeling
├── notebooks/
│   ├── Topic_CheMBL_35_20_1_data_extraction.ipynb
│   ├── Topic_CheMBL_35_20_2_feature_engineering.ipynb
│   ├── Topic_CheMBL_35_20_3_modeling.ipynb
├── src/             # Python modules for reusable code
│   ├── data/
│   │   ├── get_data.py   # Functions for reading data
│   │   ├── process_data.py # Functions for cleaning/transforming
│   ├── features/
│   │   ├── build_features.py # Functions for feature extraction
│   ├── models/
│   │   ├── train_model.py  # Training model
│   │   ├── predict_model.py # Prediction
│   │   ├── evaluate_model.py # Evaluation
```

```
├── models/       # Saved model files
├── reports/      # Generated analysis reports (optional)
├── requirements.txt # Dependencies
├── README.md
└── ...
```

3. Addressing Errors

- **SQL Error: `ERROR: operator does not exist: numeric ~ unknown`** This error occurs because the ~ operator (regular expression match) is being used on a numeric column. You're trying to check if `act.standard_value` matches a regular expression for numbers, but PostgreSQL is complaining because `act.standard_value` is a `numeric` type. The fix is to cast the numeric column to text before applying the regex.

- **scikit-learn Error: `squared=False`:** This means you're using an older version of scikit-learn. The simplest fix is to update your scikit-learn installation: `pip install scikit-learn --upgrade`. Alternatively, if you *cannot* upgrade, you can calculate the RMSE manually: `rmse = np.sqrt(mean_squared_error(y_true, y_pred))`.

4. SQL Code (Extracting 100 Rows)

```sql
-- File: data/raw/activity_data.csv

SELECT
    act.molregno,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    act.pchembl_value,
    md.chembl_id,
    cs.canonical_smiles
FROM
    activities act
JOIN molecule_dictionary md ON act.molregno = md.molregno
JOIN compound_structures cs ON md.molregno = cs.molregno
WHERE act.standard_type = 'IC50'
AND act.standard_units = 'nM'
AND act.standard_value IS NOT NULL
AND act.standard_value::text ~ '^[0-9\.]+$' -- Corrected regex and type casting
AND act.pchembl_value IS NOT NULL
LIMIT 100;
```

- **Explanation:**
    - The query joins the `activities`, `molecule_dictionary`, and `compound_structures` tables to get activity data, ChEMBL ID, and SMILES strings.
    - `WHERE` clause filters for `IC50` activity values in `nM`, ensures values are not null and use type casting and a more robust regular expression for number validation.
    - `LIMIT 100` restricts the result set to 100 rows.
    - The result is saved to `data/raw/activity_data.csv` using pgAdmin after executing the query.

5. Python Code (Jupyter Notebook - `notebooks/Topic_CheMBL_35_20_2_feature_engineering.ipynb`)

```python
# Import necessary libraries
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
```

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Define base path using os for AIMLops compatibility
base_path = os.path.abspath(os.path.join(os.getcwd(), '..'))  # Adjust if your
notebook location differs

# 1. Load data
data_path = os.path.join(base_path, 'data', 'raw', 'activity_data.csv')
df = pd.read_csv(data_path)

# 2. Data Cleaning and Preprocessing
df = df.dropna(subset=['canonical_smiles', 'pchembl_value']) # Drop rows with missing
SMILES or pChEMBL values
df = df[df['standard_value'] > 0] # Remove rows with standard_value <= 0 (log
transform won't work)

# 3. Feature Engineering with RDKit
def calculate_molecular_weight(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol:
        return Descriptors.MolWt(mol)
    else:
        return None

df['molecular_weight'] = df['canonical_smiles'].apply(calculate_molecular_weight)

# 4. Prepare data for modeling
X = df[['molecular_weight']].fillna(df['molecular_weight'].mean())  # Handle missing
molecular weight with mean imputation
y = df['pchembl_value']

# 5. Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 6. Train a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# 7. Make predictions on the test set
y_pred = model.predict(X_test)

# 8. Evaluate the model
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'Root Mean Squared Error: {rmse}')
print(f'R-squared: {r2}')
```

Key Improvements & Explanations:

- **AIMLops Path Handling:** The code uses `os.path.join(base_path, ...)` to construct file paths. This ensures the code will work regardless of the current working directory, making it

more robust within the AIMLops environment. You may need to adjust the `base_path` calculation depending on your notebook's location relative to the project root.

- **Error Handling:** The code includes `df.dropna()` to handle missing SMILES or pChEMBL values. It also includes `.fillna(df['molecular_weight'].mean())` to handle missing molecular weight.
- **Clarity and Comments:** The code is well-commented to explain each step.
- **Modularization (Ideal):** While the example is in a notebook, ideally, you would move the `calculate_molecular_weight` function, the data loading, and the model training/evaluation code into separate Python modules within the `src` directory (e.g., `src/features/build_features.py`, `src/models/train_model.py`, `src/data/get_data.py`). The notebook would then just call these functions. This makes the code more reusable and maintainable.

6. Five Examples

Example 1: Calculating LogP

```
from rdkit.Chem import AllChem
from rdkit.Chem import Crippen

def calculate_logp(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol:
        return Crippen.MolLogP(mol)
    else:
        return None

df['logp'] = df['canonical_smiles'].apply(calculate_logp)
```

Example 2: Calculating Number of Hydrogen Bond Donors

```
from rdkit.Chem import Descriptors

def calculate_hbd(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol:
        return Descriptors.NumHDonors(mol)
    else:
        return None

df['hbd'] = df['canonical_smiles'].apply(calculate_hbd)
```

Example 3: Calculating Number of Hydrogen Bond Acceptors

```
from rdkit.Chem import Descriptors

def calculate_hba(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol:
        return Descriptors.NumHAcceptors(mol)
    else:
        return None

df['hba'] = df['canonical_smiles'].apply(calculate_hba)
```

Example 4: Calculating TPSA (Topological Polar Surface Area)

```
from rdkit.Chem import Descriptors

def calculate_tpsa(smiles):
```

```
    mol = Chem.MolFromSmiles(smiles)
    if mol:
        return Descriptors.TPSA(mol)
    else:
        return None

df['tpsa'] = df['canonical_smiles'].apply(calculate_tpsa)
```

Example 5: Calculating Morgan Fingerprint (ECFP4)

```
from rdkit.Chem import AllChem
import numpy as np

def calculate_morgan_fingerprint(smiles, radius=2, nBits=2048):
    mol = Chem.MolFromSmiles(smiles)
    if mol:
        fp = AllChem.GetMorganFingerprintAsBitVect(mol, radius, nBits=nBits)
        return np.array(list(fp.ToBitString()), dtype=int) # Convert to NumPy array
    else:
        return None

# Apply the function to create a new column of fingerprints
df['morgan_fp'] = df['canonical_smiles'].apply(calculate_morgan_fingerprint)

# Flatten the fingerprint column into separate features (important for modeling)
fp_df = pd.DataFrame(df['morgan_fp'].tolist(), index=df.index)
df = pd.concat([df, fp_df], axis=1)
df.drop('morgan_fp', axis=1, inplace=True)  # Remove the original fingerprint column
```

- **Important Notes on Morgan Fingerprints:**
  - The code converts the fingerprint to a NumPy array and then flattens it into separate columns. This is *essential* for using the fingerprint as features in machine learning models.
  - The fingerprint dimensions (`nBits`) and radius are parameters that can be tuned.
  - Morgan fingerprints are widely used in cheminformatics.

Remember to install the necessary packages:

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

Vietnamese Analysis & Guidance:

1. Phân tích Mô hình và Quy trình:

Quy trình tổng thể bao gồm các bước sau:

- **Trích xuất Dữ liệu (SQL):** Trích xuất dữ liệu liên quan từ cơ sở dữ liệu Chembl 35 bằng truy vấn SQL. Điều này bao gồm lọc dữ liệu dựa trên giá trị hoạt động, thông tin mục tiêu và các tiêu chí liên quan khác. Bạn sẽ lưu dữ liệu này dưới dạng tệp CSV trong thư mục `data` của bạn.
- **Tải và Tiền Xử lý Dữ liệu (Python):** Tải dữ liệu CSV vào pandas DataFrames. Làm sạch và tiền xử lý dữ liệu, xử lý các giá trị bị thiếu, chuyển đổi kiểu dữ liệu và có khả năng áp dụng các phép biến đổi.
- **Kỹ thuật Đặc trưng (Python/RDKit):** Tạo các đặc trưng phân tử có liên quan bằng RDKit. Điều này có thể bao gồm các thuộc tính như trọng lượng phân tử, LogP, số lượng chất cho/nhận liên kết hydro, diện tích bề mặt cực topo (TPSA) và các dấu vân tay khác nhau (ví dụ: dấu vân tay Morgan).
- **Phân tích và Mô hình hóa Dữ liệu (Python):** Thực hiện phân tích dữ liệu thăm dò (EDA) để hiểu mối quan hệ giữa các đặc trưng và hoạt động. Xây dựng các mô hình dự đoán (ví dụ: hồi quy tuyến tính, rừng ngẫu nhiên, v.v.) để dự đoán hoạt động dựa trên các đặc trưng phân tử.

- **Đánh giá:** Đánh giá hiệu suất của các mô hình bằng các số liệu phù hợp (ví dụ: RMSE, R-squared).

2. Cấu trúc Mã và Tích hợp AIMLops:

Bạn đang sử dụng mẫu AIMLOps. Tuyệt vời! Điều này thường có một cấu trúc như:

```
project_root/
├── data/
│   ├── raw/         # Dữ liệu thô từ SQL (tệp CSV)
│   ├── processed/   # Dữ liệu đã xử lý sẵn sàng cho mô hình hóa
├── notebooks/
│   ├── Topic_CheMBL_35_20_1_data_extraction.ipynb
│   ├── Topic_CheMBL_35_20_2_feature_engineering.ipynb
│   ├── Topic_CheMBL_35_20_3_modeling.ipynb
├── src/            # Các mô-đun Python cho mã có thể tái sử dụng
│   ├── data/
│   │   ├── get_data.py   # Các hàm để đọc dữ liệu
│   │   ├── process_data.py # Các hàm để làm sạch/biến đổi
│   ├── features/
│   │   ├── build_features.py # Các hàm để trích xuất đặc trưng
│   ├── models/
│   │   ├── train_model.py  # Đào tạo mô hình
│   │   ├── predict_model.py # Dự đoán
│   │   ├── evaluate_model.py # Đánh giá
├── models/       # Các tệp mô hình đã lưu
├── reports/      # Các báo cáo phân tích được tạo (tùy chọn)
├── requirements.txt # Các phụ thuộc
├── README.md
└── ...
```

3. Giải quyết Lỗi:

- **Lỗi SQL:** `ERROR: operator does not exist: numeric ~ unknown` Lỗi này xảy ra vì toán tử `~` (khớp biểu thức chính quy) đang được sử dụng trên một cột số. Bạn đang cố gắng kiểm tra xem `act.standard_value` có khớp với biểu thức chính quy cho số hay không, nhưng PostgreSQL đang phàn nàn vì `act.standard_value` là kiểu `numeric`. Cách khắc phục là chuyển đổi cột số thành văn bản trước khi áp dụng regex.

- **Lỗi scikit-learn:** `squared=False`: Điều này có nghĩa là bạn đang sử dụng phiên bản cũ hơn của scikit-learn. Cách khắc phục đơn giản nhất là cập nhật cài đặt scikit-learn của bạn: `pip install scikit-learn --upgrade`. Ngoài ra, nếu bạn *không thể* nâng cấp, bạn có thể tính RMSE thủ công: `rmse = np.sqrt(mean_squared_error(y_true, y_pred))`.

4. Mã SQL (Trích xuất 100 Hàng):

```sql
-- File: data/raw/activity_data.csv

SELECT
    act.molregno,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    act.pchembl_value,
    md.chembl_id,
    cs.canonical_smiles
FROM
    activities act
JOIN molecule_dictionary md ON act.molregno = md.molregno
JOIN compound_structures cs ON md.molregno = cs.molregno
WHERE act.standard_type = 'IC50'
```

```
AND act.standard_units = 'nM'
AND act.standard_value IS NOT NULL
AND act.standard_value::text ~ '^[0-9\.]+$' -- Đã sửa regex và chuyển đổi kiểu
AND act.pchembl_value IS NOT NULL
LIMIT 100;
```

- **Giải thích:**
  - Truy vấn kết hợp các bảng `activities`, `molecule_dictionary` và `compound_structures` để lấy dữ liệu hoạt động, ID ChEMBL và chuỗi SMILES.
  - Mệnh đề `WHERE` lọc các giá trị hoạt động `IC50` trong `nM`, đảm bảo các giá trị không null và sử dụng chuyển đổi kiểu và một biểu thức chính quy mạnh mẽ hơn để xác thực số.
  - `LIMIT 100` giới hạn tập kết quả thành 100 hàng.
  - Kết quả được lưu vào `data/raw/activity_data.csv` bằng pgAdmin sau khi thực hiện truy vấn.

5. Mã Python (Jupyter Notebook - `notebooks/Topic_CheMBL_35_20_2_feature_engineering.ipynb`):

(Mã Python giống như phần tiếng Anh, chỉ cần đảm bảo rằng bạn đã cài đặt các thư viện cần thiết: `pip install pandas rdkit scikit-learn numpy`)

6. Năm Ví dụ:

(Các ví dụ tương tự như phần tiếng Anh. Hãy nhớ cài đặt các thư viện cần thiết)

Các Cải tiến và Giải thích Chính:

- **Xử lý Đường dẫn AIMLops:** Mã sử dụng `os.path.join(base_path, ...)` để xây dựng đường dẫn tệp. Điều này đảm bảo mã sẽ hoạt động bất kể thư mục làm việc hiện tại, làm cho nó mạnh mẽ hơn trong môi trường AIMLops. Bạn có thể cần điều chỉnh tính toán `base_path` tùy thuộc vào vị trí của sổ ghi chép của bạn so với thư mục gốc của dự án.
- **Xử lý Lỗi:** Mã bao gồm `df.dropna()` để xử lý các giá trị SMILES hoặc pChEMBL bị thiếu. Nó cũng bao gồm `.fillna(df['molecular_weight'].mean())` để xử lý trọng lượng phân tử bị thiếu.
- **Rõ ràng và Nhận xét:** Mã được nhận xét kỹ để giải thích từng bước.
- **Mô-đun hóa (Lý tưởng):** Mặc dù ví dụ nằm trong sổ ghi chép, nhưng lý tưởng nhất là bạn sẽ di chuyển hàm `calculate_molecular_weight`, tải dữ liệu và mã đào tạo/đánh giá mô hình vào các mô-đun Python riêng biệt trong thư mục `src` (ví dụ: `src/features/build_features.py`, `src/models/train_model.py`, `src/data/get_data.py`). Sau đó, sổ ghi chép sẽ chỉ gọi các hàm này. Điều này làm cho mã có thể tái sử dụng và dễ bảo trì hơn.

I hope this comprehensive analysis and code examples are helpful! Let me know if you have any other questions. Good luck with your project!