PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: Làm giàu target từ bộ gene

Tuyệt vời! Tôi sẽ giúp bạn xây dựng pipeline phân tích dữ liệu ChEMBL 35 kết hợp RDKit, tuân thủ theo template AIMLOps và khắc phục các lỗi bạn đã gặp phải. Chúng ta sẽ tập trung vào mã chủ đề `Topic_CheMBL_35_36`.

1. Phân tích Mô hình Phân tích (Analysis Model):

Mục tiêu của chúng ta là sử dụng dữ liệu từ ChEMBL 35 để xây dựng mô hình dự đoán hoạt tính của các hợp chất hóa học. Dưới đây là một mô hình phân tích đề xuất:

- **Thu thập và Tiền xử lý Dữ liệu (Data Acquisition and Preprocessing):**
  - Kết nối đến cơ sở dữ liệu ChEMBL 35 bằng PostgreSQL.
  - Sử dụng SQL để truy vấn và trích xuất dữ liệu cần thiết (ví dụ: cấu trúc hóa học, giá trị hoạt tính, thông tin mục tiêu).
  - Làm sạch dữ liệu:
    - Xử lý các giá trị thiếu (missing values).
    - Loại bỏ các bản ghi trùng lặp (duplicate records).
    - Chuẩn hóa các giá trị hoạt tính (ví dụ: chuyển đổi IC50, Ki, EC50 về pIC50).
    - Lọc dữ liệu để lấy 100 dòng đầu tiên cho mục đích thử nghiệm.
- **Tính toán Đặc trưng (Feature Engineering):**
  - Sử dụng RDKit để tính toán các đặc trưng phân tử từ cấu trúc hóa học (SMILES):
    - Các đặc trưng hình thái (e.g., molecular weight, logP).
    - Các đặc trưng cấu trúc (e.g., số vòng, số nguyên tử).
    - Các fingerprint (e.g., Morgan fingerprints, MACCS keys).
- **Lựa chọn Mô hình (Model Selection):**
  - Xem xét các mô hình học máy phù hợp với bài toán dự đoán hoạt tính:
    - **Hồi quy tuyến tính (Linear Regression):** Đơn giản, dễ giải thích, nhưng có thể không đủ mạnh để nắm bắt các mối quan hệ phi tuyến.
    - **Hồi quy Ridge/Lasso (Ridge/Lasso Regression):** Thêm регуляризация để tránh overfitting.
    - **Máy vector hỗ trợ (Support Vector Machines - SVM):** Hiệu quả trong không gian đặc trưng nhiều chiều.
    - **Rừng ngẫu nhiên (Random Forest):** Mạnh mẽ, có thể xử lý dữ liệu phi tuyến, nhưng khó giải thích hơn.
    - **Mạng nơ-ron (Neural Networks):** Linh hoạt, có thể học các mối quan hệ phức tạp, nhưng đòi hỏi nhiều dữ liệu và thời gian huấn luyện.
- **Đánh giá Mô hình (Model Evaluation):**
  - Chia dữ liệu thành tập huấn luyện (training set) và tập kiểm tra (test set).
  - Sử dụng các độ đo phù hợp để đánh giá hiệu suất của mô hình:
    - **Mean Squared Error (MSE):** Trung bình của bình phương sai số.
    - **R-squared (R^2):** Tỷ lệ phương sai được giải thích bởi mô hình.
    - **Root Mean Squared Error (RMSE):** Căn bậc hai của MSE, dễ diễn giải hơn.
    - **Area Under the ROC Curve (AUC):** Sử dụng nếu bài toán là phân loại (ví dụ: hoạt tính/không hoạt tính).
- **Triển khai và Giám sát (Deployment and Monitoring):**

- Triển khai mô hình đã huấn luyện để dự đoán hoạt tính của các hợp chất mới.
- Giám sát hiệu suất của mô hình theo thời gian và tái huấn luyện khi cần thiết.

2. Hướng dẫn Song ngữ (Bilingual Instructions):

English:

This project aims to build a predictive model for compound activity using ChEMBL 35 data and RDKit. We will follow the AIMLOps template, addressing the errors encountered.

Vietnamese:

Dự án này nhằm mục đích xây dựng mô hình dự đoán hoạt tính của hợp chất sử dụng dữ liệu ChEMBL 35 và RDKit. Chúng ta sẽ tuân theo template AIMLOps, giải quyết các lỗi đã gặp phải.

3. Code SQL & Python (SQL & Python Code):

SQL:

```sql
-- English: Extracting data from ChEMBL 35 (first 100 rows)
-- Vietnamese: Trích xuất dữ liệu từ ChEMBL 35 (100 dòng đầu tiên)
SELECT
    act.molregno,
    act.standard_value,
    act.standard_units,
    act.assay_id,
    md.chembl_id,
    md.structure
FROM
    activities act
JOIN
    molecule_dictionary md ON act.molregno = md.molregno
WHERE
    act.standard_type = 'IC50'
    AND act.standard_units = 'nM'
    -- Fix: Use LIKE for string matching
    AND act.standard_value LIKE '^[0-9.]+$'
LIMIT 100;
```

Python:

```python
# English: Python code for data preprocessing and feature extraction
# Vietnamese: Mã Python để tiền xử lý dữ liệu và trích xuất đặc trưng

import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import AllChem
import psycopg2
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Define the base path for the project
base_path = os.getcwd()  # Current working directory

# Database credentials
db_ip = '192.168.206.136'
db_user = 'rd'
db_pass = 'rd'
db_name = 'chembl_35'
```

```
# Function to connect to the database
def connect_to_db(ip, user, password, database):
    conn = psycopg2.connect(host=ip, user=user, password=password, database=database)
    return conn

# Function to execute SQL query and load data into pandas DataFrame
def execute_sql_query(conn, query):
    df = pd.read_sql_query(query, conn)
    return df

# Connect to the database
conn = connect_to_db(db_ip, db_user, db_pass, db_name)

# SQL query to extract data (limited to 100 rows for demonstration)
sql_query = """
SELECT
    act.molregno,
    act.standard_value,
    act.standard_units,
    act.assay_id,
    md.chembl_id,
    md.structure
FROM
    activities act
JOIN
    molecule_dictionary md ON act.molregno = md.molregno
WHERE
    act.standard_type = 'IC50'
    AND act.standard_units = 'nM'
    AND act.standard_value LIKE '^[0-9.]+$'
LIMIT 100;
"""

# Execute the SQL query and load data into a pandas DataFrame
df = execute_sql_query(conn, sql_query)

# Close the database connection
conn.close()

# Data Preprocessing
# Convert standard_value to numeric, handling errors
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')
df = df.dropna(subset=['standard_value'])

# Convert IC50 to pIC50
df['pIC50'] = -np.log10(df['standard_value'] / 1e9)  # Convert nM to M

# RDKit Feature Extraction
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is not None:
        # Calculate molecular descriptors (example: MolWt, LogP)
        molwt = Chem.Descriptors.MolWt(mol)
        logp = Chem.Descriptors.MolLogP(mol)

        # Calculate Morgan Fingerprint
        morgan_fp = AllChem.GetMorganFingerprintAsBitVect(mol, radius=2, nBits=2048)
        morgan_fp_array = np.array(list(morgan_fp.ToBitString()), dtype=int)
```

```
        return molwt, logp, morgan_fp_array
    else:
        return None, None, None

# Apply descriptor calculation to each molecule
df[['molwt', 'logp', 'morgan_fp']] = df['structure'].apply(lambda x:
pd.Series(calculate_descriptors(x)))

# Drop rows where descriptor calculation failed
df = df.dropna(subset=['molwt', 'logp', 'morgan_fp'])

# Convert Morgan Fingerprint to DataFrame
morgan_fp_df = pd.DataFrame(df['morgan_fp'].tolist())

# Concatenate descriptors and Morgan Fingerprint to the DataFrame
X = pd.concat([df[['molwt', 'logp']], morgan_fp_df], axis=1)
y = df['pIC50']

# 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)

# Model Training (Linear Regression)
model = LinearRegression()
model.fit(X_train, y_train)

# Model Prediction
y_pred = model.predict(X_test)

# Model Evaluation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

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

4. Giải thích lỗi (Error Explanation):

- **`ERROR: operator does not exist: numeric ~ unknown`:** Lỗi này xảy ra vì bạn đang cố gắng sử dụng toán tử `~` (regex match) trên cột `standard_value` kiểu số (numeric). Để khắc phục, bạn có thể sử dụng `LIKE` để so sánh chuỗi hoặc ép kiểu `standard_value` sang kiểu text trước khi so sánh. Trong code SQL trên, tôi đã thay `~` bằng `LIKE`.

- **`ValueError: Parameter 'squared' is not supported for metric 'mean_squared_error'`:** Lỗi này xảy ra vì phiên bản scikit-learn bạn đang sử dụng quá cũ. Phiên bản mới hơn của scikit-learn đã bỏ tham số `squared` trong hàm `mean_squared_error`. Bạn có thể:

    1. Nâng cấp scikit-learn lên phiên bản mới nhất (khuyến nghị).
    2. Nếu không thể nâng cấp, bạn có thể tính RMSE (Root Mean Squared Error) bằng cách lấy căn bậc hai của MSE. Trong code trên, tôi đã loại bỏ `squared=False` và tính MSE. Nếu bạn muốn tính RMSE, hãy thêm dòng `rmse = np.sqrt(mse)`.

5. Ví dụ Code (Code Examples):

Ví dụ 1: Lọc dữ liệu theo khoảng giá trị hoạt tính (Filtering Data by Activity Range):

SQL:

```sql
-- English: Select compounds with IC50 between 100 and 1000 nM
-- Vietnamese: Chọn các hợp chất có IC50 nằm trong khoảng 100 đến 1000 nM
SELECT
    act.molregno,
    md.chembl_id,
    act.standard_value
FROM
    activities act
JOIN
    molecule_dictionary md ON act.molregno = md.molregno
WHERE
    act.standard_type = 'IC50'
    AND act.standard_units = 'nM'
    AND act.standard_value >= 100
    AND act.standard_value <= 1000
LIMIT 10;
```

Python:

```python
# English: Filter compounds with pIC50 greater than 7
# Vietnamese: Lọc các hợp chất có pIC50 lớn hơn 7
filtered_df = df[df['pIC50'] > 7]
print(filtered_df.head())
```

Ví dụ 2: Tính toán số lượng vòng (Calculating Ring Count):

SQL:

```sql
-- This is difficult to do directly in SQL without custom functions
-- You would typically do this in Python using RDKit
-- Vietnamese: Rất khó để thực hiện trực tiếp trong SQL mà không có hàm tùy chỉnh
-- Bạn thường thực hiện điều này trong Python bằng RDKit
```

Python:

```python
# English: Calculate the number of rings in each molecule
# Vietnamese: Tính số lượng vòng trong mỗi phân tử
from rdkit.Chem import RingInfo

def calculate_ring_count(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is not None:
        ring_info = mol.GetRingInfo()
        return ring_info.NumRings()
    else:
        return None

df['ring_count'] = df['structure'].apply(calculate_ring_count)
print(df[['chembl_id', 'ring_count']].head())
```

Ví dụ 3: Sử dụng Fingerprint khác (Using a Different Fingerprint):

SQL:

```sql
-- SQL doesn't handle fingerprints directly. This is a Python task.
-- Vietnamese: SQL không xử lý fingerprint trực tiếp. Đây là một tác vụ Python.
```

Python:

```python
# English: Using MACCS keys instead of Morgan fingerprints
# Vietnamese: Sử dụng MACCS keys thay vì Morgan fingerprints
from rdkit.Chem import MACCSkeys
```

```python
def calculate_maccs_keys(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is not None:
        maccs_fp = MACCSkeys.GenMACCSKeys(mol)
        return np.array(list(maccs_fp.ToBitString()), dtype=int)
    else:
        return None

df['maccs_fp'] = df['structure'].apply(lambda x: pd.Series(calculate_maccs_keys(x)))
df = df.dropna(subset=['maccs_fp'])
maccs_fp_df = pd.DataFrame(df['maccs_fp'].tolist())

# Now use maccs_fp_df as your feature matrix
```

Ví dụ 4: Chuẩn hóa giá trị hoạt tính (Normalizing Activity Values):

SQL:

```sql
-- This can be done in Python after fetching the data
-- Vietnamese: Điều này có thể được thực hiện trong Python sau khi lấy dữ liệu
```

Python:

```python
# English:  Normalize pIC50 values to a range of 0-1
# Vietnamese: Chuẩn hóa giá trị pIC50 về khoảng 0-1
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
df['pIC50_normalized'] = scaler.fit_transform(df[['pIC50']])
print(df[['chembl_id', 'pIC50', 'pIC50_normalized']].head())
```

Ví dụ 5: Sử dụng mô hình khác (Using a different model):

SQL:

```sql
-- SQL is not used for model training
-- Vietnamese: SQL không được sử dụng để huấn luyện mô hình
```

Python:

```python
# English: Using Random Forest Regressor instead of Linear Regression
# Vietnamese: Sử dụng Random Forest Regressor thay vì Linear Regression
from sklearn.ensemble import RandomForestRegressor

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

# Model Training (Random Forest)
model = RandomForestRegressor(n_estimators=100, random_state=42)  # You can tune
hyperparameters
model.fit(X_train, y_train)

# Model Prediction
y_pred = model.predict(X_test)

# Model Evaluation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

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

Lưu ý quan trọng:

- **Xử lý lỗi `None` từ RDKit:** Code trên đã bao gồm việc kiểm tra `None` khi tính toán descriptors và loại bỏ các hàng chứa `None`. Điều này rất quan trọng vì RDKit có thể trả về `None` nếu không thể xử lý cấu trúc SMILES.
- **Kết nối Cơ sở Dữ liệu:** Đảm bảo rằng bạn đã cài đặt thư viện `psycopg2` (`pip install psycopg2-binary`) để kết nối với PostgreSQL.
- **Đường dẫn Tệp:** Sử dụng `os.path.join(base_path, "data", "your_file.csv")` để tạo đường dẫn tệp, đảm bảo tính di động của code.
- **Phiên bản Scikit-learn:** Khuyến nghị nâng cấp scikit-learn để tận dụng các tính năng mới nhất và tránh các lỗi liên quan đến phiên bản.
- **Giới hạn Dữ liệu:** Trong ví dụ này, tôi đã giới hạn ở 100 dòng dữ liệu để thử nghiệm. Khi chạy trên toàn bộ tập dữ liệu, hãy cân nhắc việc sử dụng các kỹ thuật tối ưu hóa để cải thiện hiệu suất.
- **Tối ưu hóa Mô hình:** Các ví dụ trên chỉ là điểm khởi đầu. Hãy thử nghiệm với các mô hình khác nhau, các đặc trưng khác nhau và các kỹ thuật điều chỉnh siêu tham số để cải thiện hiệu suất dự đoán.

Chúc bạn thành công với dự án của mình! Hãy cho tôi biết nếu bạn có bất kỳ câu hỏi nào khác.