PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: Tinh chỉnh siêu tham số bằng Optuna cho mô hình thuốc

Tuyệt vời! Tôi rất vui được hỗ trợ bạn trong dự án phân tích dữ liệu ChEMBL 35 này. Với vai trò là một chuyên gia nghiên cứu và phát triển dược học, tôi sẽ cung cấp cho bạn các phân tích, hướng dẫn, code mẫu (SQL và Python) và các ví dụ cụ thể để bạn có thể triển khai dự án một cách hiệu quả nhất.

Tổng quan về dự án:

Dự án của bạn tập trung vào việc phân tích dữ liệu từ cơ sở dữ liệu ChEMBL 35, kết hợp với thư viện RDKit để hỗ trợ các nghiên cứu và phát triển thuốc. Việc sử dụng cấu trúc thư mục theo chuẩn MLOps sẽ giúp bạn tổ chức và quản lý dự án một cách chuyên nghiệp.

Phân tích mô hình:

Mục tiêu chính của bạn có thể là:

1. **Khám phá dữ liệu (Exploratory Data Analysis - EDA):**
    - Hiểu rõ cấu trúc và nội dung của dữ liệu ChEMBL 35.
    - Xác định các thuộc tính quan trọng và mối quan hệ giữa chúng.
    - Tìm kiếm các xu hướng và mẫu tiềm năng có thể giúp ích cho việc phát triển thuốc.
2. **Xây dựng mô hình dự đoán (Predictive Modeling):**
    - Dự đoán hoạt tính sinh học của các phân tử dựa trên cấu trúc hóa học của chúng.
    - Xác định các đặc điểm cấu trúc quan trọng ảnh hưởng đến hoạt tính.
    - Tối ưu hóa cấu trúc phân tử để cải thiện hoạt tính.
3. **Phân tích QSAR/QSPR (Quantitative Structure-Activity Relationship/Quantitative Structure-Property Relationship):**
    - Xây dựng mô hình liên hệ giữa cấu trúc hóa học và hoạt tính sinh học/tính chất vật lý của phân tử.
    - Sử dụng mô hình để dự đoán hoạt tính/tính chất của các phân tử mới.
    - Tìm kiếm các phân tử có tiềm năng trở thành thuốc.

Hướng dẫn song ngữ:

1. Kết nối đến cơ sở dữ liệu ChEMBL 35 (Connecting to ChEMBL 35 Database):

- **Tiếng Việt:** Sử dụng thư viện `psycopg2` trong Python để kết nối đến cơ sở dữ liệu PostgreSQL.
- **English:** Use the `psycopg2` library in Python to connect to the PostgreSQL database.

```python
import psycopg2

# Database credentials
db_params = {
    'host': '192.168.206.136',
    'user': 'rd',
    'password': 'rd',
    'database': 'chembl_35'
}

# Function to connect to the database
def connect_to_db(params):
    try:
        conn = psycopg2.connect(**params)
```

```
        print("Connected to the database successfully!")
        return conn
    except psycopg2.Error as e:
        print(f"Error connecting to the database: {e}")
        return None

# Establish connection
conn = connect_to_db(db_params)
```

2. Truy vấn dữ liệu từ cơ sở dữ liệu (Querying Data from the Database):

- **Tiếng Việt:** Sử dụng SQL để truy vấn dữ liệu từ các bảng trong cơ sở dữ liệu ChEMBL 35.
- **English:** Use SQL to query data from the tables in the ChEMBL 35 database.

```
-- SQL query to retrieve data from the 'activities' and 'molecule_dictionary' tables
-- Retrieving only 100 records for demonstration purposes
SELECT act.molregno, md.chembl_id, act.standard_type, act.standard_value,
act.standard_units
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 IS NOT NULL
  AND act.standard_value > 0
LIMIT 100;
```

3. Tiền xử lý dữ liệu (Data Preprocessing):

- **Tiếng Việt:** Làm sạch và chuẩn hóa dữ liệu để đảm bảo chất lượng và tính nhất quán. Loại bỏ các giá trị thiếu, xử lý các giá trị ngoại lệ và chuyển đổi dữ liệu về định dạng phù hợp.
- **English:** Clean and normalize the data to ensure quality and consistency. Remove missing values, handle outliers, and convert data to the appropriate format.

```
import pandas as pd
import numpy as np

# Function to load data from a CSV file
def load_data(file_path):
    try:
        data = pd.read_csv(file_path)
        print(f"Data loaded successfully from {file_path}")
        return data
    except FileNotFoundError:
        print(f"Error: File not found at {file_path}")
        return None

# Function to clean the data
def clean_data(df):
    # Remove missing values
    df = df.dropna()
    # Remove duplicate rows
    df = df.drop_duplicates()
    # Convert 'standard_value' to numeric, handling errors
    df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')
    # Remove rows where 'standard_value' is NaN after conversion
    df = df.dropna(subset=['standard_value'])
    # Filter out non-positive values
    df = df[df['standard_value'] > 0]
    print("Data cleaning complete.")
    return df

# Example usage:
```

```
# Assuming base_path is defined and the CSV file is in the data directory
base_path = '.'  # Replace with your actual base path
csv_file_path = os.path.join(base_path, 'data', 'your_data_file.csv')  # Replace with
your actual file name
data = load_data(csv_file_path)

if data is not None:
    cleaned_data = clean_data(data)
    print(cleaned_data.head())
```

4. Tính toán đặc trưng phân tử (Molecular Feature Calculation):

- **Tiếng Việt:** Sử dụng RDKit để tính toán các đặc trưng hóa học của phân tử, như trọng lượng phân tử, độ tan, số lượng liên kết, v.v.
- **English:** Use RDKit to calculate molecular features such as molecular weight, solubility, number of bonds, etc.

```
from rdkit import Chem
from rdkit.Chem import Descriptors

# Function to calculate molecular descriptors using RDKit
def calculate_descriptors(smiles):
    try:
        mol = Chem.MolFromSmiles(smiles)
        if mol is None:
            return None  # Handle invalid SMILES strings

        descriptors = {}
        descriptors['MolWt'] = Descriptors.MolWt(mol)
        descriptors['LogP'] = Descriptors.MolLogP(mol)
        descriptors['NumHAcceptors'] = Descriptors.NumHAcceptors(mol)
        descriptors['NumHDonors'] = Descriptors.NumHDonors(mol)
        return descriptors
    except Exception as e:
        print(f"Error calculating descriptors for SMILES {smiles}: {e}")
        return None

# Example usage:
# Assuming you have a DataFrame with a 'smiles' column
def add_descriptors_to_df(df, smiles_column='smiles'):
    # Apply the descriptor calculation to each SMILES in the DataFrame
    df['descriptors'] = df[smiles_column].apply(calculate_descriptors)

    # Expand the 'descriptors' column into separate columns
    df = pd.concat([df, df['descriptors'].apply(pd.Series)], axis=1)

    # Remove the original 'descriptors' column
    df = df.drop(columns=['descriptors'])

    return df

# Example of how to use this function (assuming 'cleaned_data' DataFrame is available)
# Ensure that 'cleaned_data' DataFrame has a 'smiles' column
# cleaned_data = add_descriptors_to_df(cleaned_data, smiles_column='smiles')
# print(cleaned_data.head())
```

5. Xây dựng mô hình học máy (Machine Learning Model Building):

- **Tiếng Việt:** Sử dụng các thuật toán học máy để xây dựng mô hình dự đoán hoạt tính sinh học của phân tử.

- **English:** Use machine learning algorithms to build a model for predicting the biological activity of molecules.

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Function to train a machine learning model
def train_model(df, features, target):
    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(df[features], df[target],
test_size=0.2, random_state=42)

    # Initialize the model
    model = RandomForestRegressor(n_estimators=100, random_state=42)

    # Train the model
    model.fit(X_train, y_train)

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

    # Evaluate the model
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print(f"Mean Squared Error: {mse}")
    print(f"R^2 Score: {r2}")

    return model, X_test, y_test, y_pred

# Example usage:
# Assuming you have a DataFrame with calculated descriptors and activity values
# Ensure that 'cleaned_data' DataFrame has the necessary columns for features and
target
# Define the features and target
# features = ['MolWt', 'LogP', 'NumHAcceptors', 'NumHDonors']  # Replace with your
feature columns
# target = 'standard_value'  # Replace with your target column

# Train the model
# model, X_test, y_test, y_pred = train_model(cleaned_data, features, target)
```

Sửa lỗi:

a. ERROR: operator does not exist: numeric ~ unknown:

- **Tiếng Việt:** Lỗi này xảy ra do bạn đang cố gắng sử dụng toán tử ~ (tương tự như LIKE trong SQL) để so sánh một cột số (numeric) với một chuỗi (unknown). Để khắc phục, bạn cần đảm bảo rằng cả hai vế của toán tử ~ đều là chuỗi, hoặc sử dụng các toán tử so sánh số học phù hợp (ví dụ: =, >, <). Trong trường hợp này, bạn có thể loại bỏ điều kiện `act.standard_value ~ '^[0-9\.]+$'` vì nó không cần thiết.
- **English:** This error occurs because you are trying to use the ~ operator (similar to LIKE in SQL) to compare a numeric column with a string. To fix this, ensure that both sides of the ~ operator are strings, or use appropriate numeric comparison operators (e.g., =, >, <). In this case, you can remove the condition `act.standard_value ~ '^[0-9\.]+$'` as it is not necessary.

```
-- Corrected SQL query
SELECT act.molregno, md.chembl_id, act.standard_type, act.standard_value,
```

```
act.standard_units
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 IS NOT NULL
  AND act.standard_value > 0
LIMIT 100;
```

b. Phiên bản scikit-learn cũ không hỗ trợ tham số squared=False trong hàm mean_squared_error:

- **Tiếng Việt:** Nếu bạn đang sử dụng phiên bản scikit-learn cũ, hãy loại bỏ tham số `squared=False` khỏi hàm `mean_squared_error`. Nếu bạn muốn tính căn bậc hai của MSE (RMSE), bạn có thể sử dụng hàm `np.sqrt()` để tính toán sau.
- **English:** If you are using an older version of scikit-learn, remove the `squared=False` parameter from the `mean_squared_error` function. If you want to calculate the square root of MSE (RMSE), you can use the `np.sqrt()` function to calculate it afterwards.

```
from sklearn.metrics import mean_squared_error
import numpy as np

# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)

# Calculate Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)

print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}")
```

5 Ví dụ code SQL và Python mẫu:

Ví dụ 1: Lọc các phân tử có hoạt tính IC50 dưới 100nM (Filtering molecules with IC50 activity below 100nM):

- **SQL:**

```
SELECT 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
LIMIT 100;
```

- **Python:**

```
import pandas as pd
import psycopg2

# Database credentials
db_params = {
    'host': '192.168.206.136',
    'user': 'rd',
    'password': 'rd',
    'database': 'chembl_35'
}

# SQL query to retrieve data from the 'activities' and 'molecule_dictionary' tables
sql_query = """
SELECT 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
LIMIT 100;
"""

# Function to execute the SQL query and return the data as a DataFrame
def execute_query(db_params, sql_query):
    try:
        # Establish a connection to the PostgreSQL database
        conn = psycopg2.connect(**db_params)

        # Create a cursor object
        cur = conn.cursor()

        # Execute the SQL query
        cur.execute(sql_query)

        # Fetch all the results
        results = cur.fetchall()

        # Get column names from the cursor description
        column_names = [desc[0] for desc in cur.description]

        # Convert the results into a pandas DataFrame
        df = pd.DataFrame(results, columns=column_names)

        # Close the cursor and connection
        cur.close()
        conn.close()

        print("Query executed successfully!")
        return df
    except psycopg2.Error as e:
        print(f"Error executing query: {e}")
        return None

# Execute the query and store the results in a DataFrame
df = execute_query(db_params, sql_query)

# Print the DataFrame
if df is not None:
    print(df.head())
```

Ví dụ 2: Tính trọng lượng phân tử trung bình (Calculating average molecular weight):

- **SQL:**

```
SELECT AVG(md.molecular_weight) AS average_molecular_weight
FROM molecule_dictionary md
LIMIT 100;
```

- **Python:**

```
import pandas as pd
import psycopg2

# Database credentials
db_params = {
    'host': '192.168.206.136',
    'user': 'rd',
```

```
    'password': 'rd',
    'database': 'chembl_35'
}

# SQL query to retrieve data from the 'activities' and 'molecule_dictionary' tables
sql_query = """
SELECT md.molecular_weight
FROM molecule_dictionary md
LIMIT 100;
"""

# Function to execute the SQL query and return the data as a DataFrame
def execute_query(db_params, sql_query):
    try:
        # Establish a connection to the PostgreSQL database
        conn = psycopg2.connect(**db_params)

        # Create a cursor object
        cur = conn.cursor()

        # Execute the SQL query
        cur.execute(sql_query)

        # Fetch all the results
        results = cur.fetchall()

        # Get column names from the cursor description
        column_names = [desc[0] for desc in cur.description]

        # Convert the results into a pandas DataFrame
        df = pd.DataFrame(results, columns=column_names)

        # Close the cursor and connection
        cur.close()
        conn.close()

        print("Query executed successfully!")
        return df
    except psycopg2.Error as e:
        print(f"Error executing query: {e}")
        return None

# Execute the query and store the results in a DataFrame
df = execute_query(db_params, sql_query)
if df is not None:
    average_molecular_weight = df['molecular_weight'].mean()
    print(f"Average Molecular Weight: {average_molecular_weight}")
```

Ví dụ 3: Tìm các phân tử có chứa vòng benzen (Finding molecules containing a benzene ring):

- **SQL:**

```
-- This SQL query will not directly identify molecules with a benzene ring.
-- You would typically use substructure searching capabilities within ChEMBL's
interface or RDKit.
-- This is a placeholder and might require a different approach using ChEMBL's API or
RDKit integration.
SELECT md.chembl_id
FROM molecule_dictionary md
WHERE md.molecule_structures LIKE '%c1ccccc1%'  -- This is a simplistic approach and
```

```
might not be accurate.
LIMIT 100;
```

- **Python:**

```
from rdkit import Chem
from rdkit.Chem import AllChem
import pandas as pd
import psycopg2

# Database credentials
db_params = {
    'host': '192.168.206.136',
    'user': 'rd',
    'password': 'rd',
    'database': 'chembl_35'
}

# Function to execute the SQL query and return the data as a DataFrame
def execute_query(db_params, sql_query):
    try:
        # Establish a connection to the PostgreSQL database
        conn = psycopg2.connect(**db_params)

        # Create a cursor object
        cur = conn.cursor()

        # Execute the SQL query
        cur.execute(sql_query)

        # Fetch all the results
        results = cur.fetchall()

        # Get column names from the cursor description
        column_names = [desc[0] for desc in cur.description]

        # Convert the results into a pandas DataFrame
        df = pd.DataFrame(results, columns=column_names)

        # Close the cursor and connection
        cur.close()
        conn.close()

        print("Query executed successfully!")
        return df
    except psycopg2.Error as e:
        print(f"Error executing query: {e}")
        return None

# SQL query to retrieve data from the 'activities' and 'molecule_dictionary' tables
sql_query = """
SELECT md.chembl_id, md.molecule_structures
FROM molecule_dictionary md
LIMIT 100;
"""

# Define the SMILES pattern for a benzene ring
benzene_smiles = 'c1ccccc1'
benzene_mol = Chem.MolFromSmiles(benzene_smiles)

# Function to check if a molecule contains a benzene ring
```

```
def contains_benzene(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return False
    return mol.HasSubstructMatch(benzene_mol)

# Execute the query and store the results in a DataFrame
df = execute_query(db_params, sql_query)

# Apply the function to check for benzene ring
if df is not None:
    df['has_benzene'] = df['molecule_structures'].apply(contains_benzene)

    # Filter the DataFrame to only include molecules with a benzene ring
    benzene_molecules = df[df['has_benzene']]

    # Print the DataFrame
    print(benzene_molecules.head())
```

Ví dụ 4: Phân tích mối tương quan giữa LogP và hoạt tính (Analyzing the correlation between LogP and activity):

- **SQL:** (Không thể thực hiện trực tiếp trong SQL, cần kết hợp với Python)
- **Python:**

```
import pandas as pd
import psycopg2
from rdkit import Chem
from rdkit.Chem import Descriptors

# Database credentials
db_params = {
    'host': '192.168.206.136',
    'user': 'rd',
    'password': 'rd',
    'database': 'chembl_35'
}

# SQL query to retrieve data from the 'activities' and 'molecule_dictionary' tables
sql_query = """
SELECT md.chembl_id, md.molecule_structures, 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 IS NOT NULL
  AND act.standard_value > 0
LIMIT 100;
"""

# Function to execute the SQL query and return the data as a DataFrame
def execute_query(db_params, sql_query):
    try:
        # Establish a connection to the PostgreSQL database
        conn = psycopg2.connect(**db_params)

        # Create a cursor object
        cur = conn.cursor()
```

```
        # Execute the SQL query
        cur.execute(sql_query)

        # Fetch all the results
        results = cur.fetchall()

        # Get column names from the cursor description
        column_names = [desc[0] for desc in cur.description]

        # Convert the results into a pandas DataFrame
        df = pd.DataFrame(results, columns=column_names)

        # Close the cursor and connection
        cur.close()
        conn.close()

        print("Query executed successfully!")
        return df
    except psycopg2.Error as e:
        print(f"Error executing query: {e}")
        return None

# Function to calculate LogP using RDKit
def calculate_logp(smiles):
    try:
        mol = Chem.MolFromSmiles(smiles)
        if mol is None:
            return None
        return Descriptors.MolLogP(mol)
    except:
        return None

# Execute the query and store the results in a DataFrame
df = execute_query(db_params, sql_query)

# Calculate LogP for each molecule
if df is not None:
    df['logp'] = df['molecule_structures'].apply(calculate_logp)
    df = df.dropna(subset=['logp', 'standard_value'])

    # Calculate the correlation between LogP and activity
    correlation = df['logp'].corr(df['standard_value'])
    print(f"Correlation between LogP and IC50: {correlation}")
```

Ví dụ 5: Tạo biểu đồ phân tán giữa trọng lượng phân tử và LogP (Creating a scatter plot between molecular weight and LogP):

- **SQL:** (Không thể thực hiện trực tiếp trong SQL, cần kết hợp với Python)
- **Python:**

```
import pandas as pd
import psycopg2
from rdkit import Chem
from rdkit.Chem import Descriptors
import matplotlib.pyplot as plt

# Database credentials
db_params = {
    'host': '192.168.206.136',
```

```
    'user': 'rd',
    'password': 'rd',
    'database': 'chembl_35'
}

# SQL query to retrieve data from the 'activities' and 'molecule_dictionary' tables
sql_query = """
SELECT md.chembl_id, md.molecule_structures
FROM molecule_dictionary md
LIMIT 100;
"""

# Function to execute the SQL query and return the data as a DataFrame
def execute_query(db_params, sql_query):
    try:
        # Establish a connection to the PostgreSQL database
        conn = psycopg2.connect(**db_params)

        # Create a cursor object
        cur = conn.cursor()

        # Execute the SQL query
        cur.execute(sql_query)

        # Fetch all the results
        results = cur.fetchall()

        # Get column names from the cursor description
        column_names = [desc[0] for desc in cur.description]

        # Convert the results into a pandas DataFrame
        df = pd.DataFrame(results, columns=column_names)

        # Close the cursor and connection
        cur.close()
        conn.close()

        print("Query executed successfully!")
        return df
    except psycopg2.Error as e:
        print(f"Error executing query: {e}")
        return None

# Function to calculate molecular weight and LogP using RDKit
def calculate_mw_logp(smiles):
    try:
        mol = Chem.MolFromSmiles(smiles)
        if mol is None:
            return None, None
        mw = Descriptors.MolWt(mol)
        logp = Descriptors.MolLogP(mol)
        return mw, logp
    except:
        return None, None

# Execute the query and store the results in a DataFrame
df = execute_query(db_params, sql_query)

# Calculate molecular weight and LogP for each molecule
if df is not None:
```

```
df['mw'], df['logp'] = zip(*df['molecule_structures'].apply(calculate_mw_logp))
df = df.dropna(subset=['mw', 'logp'])

# Create a scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(df['mw'], df['logp'], alpha=0.5)
plt.xlabel('Molecular Weight')
plt.ylabel('LogP')
plt.title('Scatter Plot of Molecular Weight vs LogP')
plt.grid(True)
plt.show()
```

Lưu ý quan trọng:

- **SMILES Strings:** Đảm bảo rằng cột `molecule_structures` chứa chuỗi SMILES hợp lệ.
- **Error Handling:** Xử lý các lỗi có thể xảy ra trong quá trình tính toán đặc trưng phân tử.
- **Data Scaling:** Cân nhắc việc chuẩn hóa hoặc масштабирование dữ liệu trước khi xây dựng mô hình học máy.
- **Feature Selection:** Lựa chọn các đặc trưng quan trọng nhất để cải thiện hiệu suất của mô hình.
- **Model Evaluation:** Đánh giá mô hình bằng các mét đo phù hợp (ví dụ: MSE, RMSE, R^2).

Tôi hy vọng những hướng dẫn và ví dụ này sẽ giúp bạn triển khai dự án của mình một cách hiệu quả. Nếu bạn có bất kỳ câu hỏi nào khác, đừng ngần ngại hỏi!