PharmApp Suite

🧠 AI for Drug Discovery and Development 🍼

🧩 Topic: Predicting bioactivity from molecular descriptors

Okay, I understand. You're working on a Chembl 35 data analysis project using RDKit and need help structuring your workflow and resolving errors. You have a specific folder structure based on the AIMLops template, and you're running SQL queries on a PostgreSQL database to extract data for your Jupyter Notebook analysis. I'll provide a detailed breakdown in both English and Vietnamese, along with code examples, addressing the errors you've encountered and limiting the data extraction to 100 rows for efficiency.

Let's break this down systematically.

1. Analysis Model Overview:

The goal of your project seems to be to analyze Chembl 35 data using RDKit and scikit-learn within a structured AIMLops framework. Here's a likely workflow and what you are trying to accomplish:

- **Data Extraction (SQL):** You're using SQL queries to extract relevant data from the Chembl 35 database on your PostgreSQL server. This will likely involve joining tables like `activities`, `assays`, `molecule_dictionary`, etc. to get information about compounds, their biological activity, and their properties.
- **Data Preparation (Python/RDKit):** After extracting the data, you'll use Python and RDKit to:
  - Read the CSV data into a Pandas DataFrame.
  - Generate molecular descriptors (e.g., fingerprints, physicochemical properties) using RDKit based on the SMILES strings of the compounds.
  - Clean and preprocess the data (e.g., handle missing values, scale numerical features).
- **Model Building and Evaluation (Python/scikit-learn):**
  - Split the data into training and testing sets.
  - Train a machine learning model (e.g., regression model to predict activity values or a classification model to predict activity classes) using scikit-learn.
  - Evaluate the model's performance using appropriate metrics (e.g., R-squared, RMSE, AUC, accuracy).

2. Addressing Errors and Structuring the Code:

Let's address the errors you've mentioned and provide a clear code structure.

a. SQL Error: `ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '^[0-9\.]+$'`

This error means you're trying to use the regular expression operator `~` on a `numeric` column (`act.standard_value`). PostgreSQL's `~` operator is for string matching. To filter numeric data, you should use numeric comparison operators.

Solution: Change the query to a proper numeric range check.

b. Python Error: `old scikit-learn version does not support parameters squared=False in the mean_squared_error function`

This means you are using an older version of scikit-learn. The `squared=False` parameter was introduced in a later version to return the Root Mean Squared Error (RMSE) instead of Mean Squared Error (MSE).

Solution: 1. **Update scikit-learn:** The best solution is to update your scikit-learn version. `pip install --upgrade scikit-learn` 2. **Calculate RMSE manually:** If you cannot update scikit-learn, calculate the RMSE manually by taking the square root of the MSE.

3. Code Examples (English and Vietnamese):

Here's the code, organized according to your specifications, with explanations and error handling. I'll provide SQL for data extraction and Python for data processing, model building, and error correction.

SQL (PostgreSQL) - `Topic_CheMBL_35_2.sql`

```sql
-- Topic Code: Topic_CheMBL_35_2
-- Topic Name: (Your Topic Name - e.g., "IC50 Prediction for Target X")

-- Extracting activity data, molecule information, and limiting to 100 rows
SELECT
    act.activity_id,
    mol.molregno,
    mol.chembl_id,
    mol.pref_name,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    act.pchembl_value,
    act.assay_id
FROM
    activities act
JOIN
    molecule_dictionary mol ON act.molregno = mol.molregno
JOIN
    assays assay ON act.assay_id = assay.assay_id
WHERE
    act.standard_type = 'IC50'  -- Example: Filtering for IC50 values
    AND act.standard_value IS NOT NULL  -- Ensure standard_value is not null
    AND act.standard_value > 0     --Ensuring standard_value is positive
    AND act.standard_units = 'nM' -- Filtering for values in nM
LIMIT 100;
```

Explanation (English):

- This SQL query joins the `activities`, `molecule_dictionary`, and `assays` tables to retrieve relevant information.
- It filters for activities with `standard_type = 'IC50'` and `standard_units = 'nM'` (you can adjust these based on your specific needs).
- The `LIMIT 100` clause restricts the result set to 100 rows.
- The `WHERE` clause includes a numeric comparison `act.standard_value > 0` to avoid the numeric-string comparison error. It also checks for NULL values to prevent errors in later processing.

Explanation (Vietnamese):

- Câu lệnh SQL này kết hợp các bảng `activities`, `molecule_dictionary` và `assays` để lấy thông tin liên quan.
- Nó lọc các hoạt động với `standard_type = 'IC50'` và `standard_units = 'nM'` (bạn có thể điều chỉnh chúng dựa trên nhu cầu cụ thể của bạn).
- Mệnh đề `LIMIT 100` giới hạn kết quả trả về chỉ 100 hàng.
- Mệnh đề `WHERE` bao gồm so sánh số `act.standard_value > 0` để tránh lỗi so sánh số với chuỗi. Nó cũng kiểm tra các giá trị NULL để ngăn ngừa lỗi trong quá trình xử lý sau này.

Python (Jupyter Notebook) - `Topic_CheMBL_35_2_1_Data_Preprocessing.ipynb`

```
# Topic Code: Topic_CheMBL_35_2
# Topic Name: (Your Topic Name - e.g., "IC50 Prediction for Target X")

import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import AllChem
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
from sklearn.preprocessing import StandardScaler
import math  # Import the math module

# 1. Define Paths
base_path = "."  # Assuming the notebook is in the root of your project
data_path = os.path.join(base_path, "data", "Topic_CheMBL_35_2.csv") # Assuming the
SQL script was saved as Topic_CheMBL_35_2.csv

# 2. Load Data
try:
    df = pd.read_csv(data_path)
    print(f"Data loaded successfully from {data_path}")
except FileNotFoundError:
    print(f"Error: File not found at {data_path}.  Make sure you have run the SQL
script and saved the data correctly.")
    exit()

# 3. Data Cleaning and Preparation
# Handle missing values (replace with the mean for numerical columns)
for col in df.columns:
    if df[col].dtype == 'float64' or df[col].dtype == 'int64':
        df[col] = df[col].fillna(df[col].mean())

# Convert SMILES to RDKit Mol objects and calculate descriptors
def calculate_descriptors(smiles):
    try:
        mol = Chem.MolFromSmiles(smiles)
        if mol is not None:
            fp = AllChem.GetMorganFingerprintAsBitVect(mol, 2, nBits=1024) # You can
adjust nBits
            return list(fp)
        else:
            return None
    except:
        return None


# Assuming you have a column named 'chembl_id' in your DataFrame
# First, get the SMILES strings. You'll need to figure out how to get SMILES from the
chembl_id.
# This example uses dummy SMILES.  Replace with actual SMILES retrieval.
# For the sake of simplicity, let's create dummy SMILES.  In reality, you'd have to
fetch SMILES from ChEMBL database using the chembl_id.
dummy_smiles = ['CCO', 'C1=CC=CC=C1', 'CC(=O)O', 'c1ccccc1C(=O)O',
'CN1C=NC2=C1N=CN=C2N']  # Example SMILES

# Make sure to create a 'smiles' column
```

```
df['smiles'] = dummy_smiles * (len(df) // len(dummy_smiles)) + dummy_smiles[:len(df) %
len(dummy_smiles)]

# Apply the function to create a new column 'descriptors'
df['descriptors'] = df['smiles'].apply(calculate_descriptors)

# Drop rows where descriptor calculation failed
df = df.dropna(subset=['descriptors'])

# Convert descriptors to a NumPy array
X = np.array(df['descriptors'].tolist())

# Target variable (e.g., pChEMBL value)
y = df['pchembl_value'].values

# 4. Data Scaling
scaler = StandardScaler()
X = scaler.fit_transform(X)

# 5. Train/Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

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

# 7. Model Evaluation
y_pred = model.predict(X_test)

# Handle the scikit-learn version issue:
try:
    mse = mean_squared_error(y_test, y_pred, squared=False)  # Try the new version
first
    rmse = mse
    print("Using scikit-learn with squared=False parameter.")
except TypeError:
    print("Using older scikit-learn version. Calculating RMSE manually.")
    mse = mean_squared_error(y_test, y_pred)
    rmse = math.sqrt(mse)  # Calculate RMSE manually

r2 = r2_score(y_test, y_pred)

print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2): {r2}")
```

Explanation (English):

- **Loads Data:** Reads the CSV data extracted from the database.
- **Handles Missing Values:** Fills any missing values in numerical columns with the mean. This is a simple approach; you might need more sophisticated imputation strategies.
- **Calculates Descriptors:** Uses RDKit to generate molecular fingerprints. This part is crucial. **Important:** The code currently uses dummy SMILES. You **must** replace this with code that retrieves the actual SMILES strings from the ChEMBL database based on the `chembl_id` column. You'll likely need to use the ChEMBL API or another database query to do this.
- **Splits Data:** Splits the data into training and testing sets.
- **Trains a Linear Regression Model:** Uses scikit-learn's `LinearRegression` model. You can experiment with other models.

- **Evaluates the Model:** Calculates the RMSE and R-squared to evaluate the model's performance. The code includes a check for the scikit-learn version to handle the `squared=False` parameter correctly.

Explanation (Vietnamese):

- **Tải Dữ Liệu:** Đọc dữ liệu CSV đã được trích xuất từ cơ sở dữ liệu.
- **Xử Lý Giá Trị Thiếu:** Điền bất kỳ giá trị thiếu nào trong các cột số bằng giá trị trung bình. Đây là một phương pháp đơn giản; bạn có thể cần các chiến lược điền khuyết phức tạp hơn.
- **Tính Toán Các Thuộc Tính Phân Tử:** Sử dụng RDKit để tạo ra các fingerprint phân tử. Phần này rất quan trọng. **Quan trọng:** Mã hiện tại sử dụng SMILES giả. Bạn **phải** thay thế điều này bằng mã lấy chuỗi SMILES thực tế từ cơ sở dữ liệu ChEMBL dựa trên cột `chembl_id`. Bạn có thể cần sử dụng API ChEMBL hoặc một truy vấn cơ sở dữ liệu khác để làm điều này.
- **Chia Dữ Liệu:** Chia dữ liệu thành các tập huấn luyện và kiểm tra.
- **Huấn Luyện Mô Hình Hồi Quy Tuyến Tính:** Sử dụng mô hình `LinearRegression` của scikit-learn. Bạn có thể thử nghiệm với các mô hình khác.
- **Đánh Giá Mô Hình:** Tính toán RMSE và R-squared để đánh giá hiệu suất của mô hình. Mã này bao gồm kiểm tra phiên bản scikit-learn để xử lý tham số `squared = False` một cách chính xác.

4. Five Examples (based on topic codes)

Here are five examples of topic names/codes you could explore, and corresponding modifications to the code:

- **Example 1: Topic_CheMBL_35_2_IC50_Activity_Prediction**
  - **Topic:** Predicting IC50 activity values using molecular descriptors.
  - **SQL (Modification):** Keep the SQL as is, since it already extracts IC50 values. You might add more specific filtering for a particular target or assay.
  - **Python (Modification):** Use `pchembl_value` as the target variable (`y = df['pchembl_value'].values`). Ensure the SMILES retrieval is working correctly.
- **Example 2: Topic_CheMBL_35_2_Ki_Activity_Prediction**
  - **Topic:** Predicting Ki activity values using molecular descriptors.
  - **SQL (Modification):** `WHERE act.standard_type = 'Ki'` (and adjust `standard_units` accordingly).
  - **Python (Modification):** Same as Example 1, but using the Ki values from the SQL output.
- **Example 3: Topic_CheMBL_35_2_LogP_Correlation**
  - **Topic:** Investigating the correlation between calculated LogP and activity.
  - **SQL (Modification):** No change to the SQL structure. Keep fetching the relevant compounds and activity data.
  - **Python (Modification):**
    - Use RDKit to calculate LogP for each molecule:

```
from rdkit.Chem import Descriptors
def calculate_logp(smiles):
    try:
        mol = Chem.MolFromSmiles(smiles)
        if mol is not None:
            return Descriptors.MolLogP(mol)
        else:
            return None
    except:
        return None

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

```
df = df.dropna(subset=['logp'])  # Drop rows where LogP couldn't be calculated
```

- Analyze the correlation between the calculated `logp` and `pchembl_value`. You might perform linear regression or calculate correlation coefficients using Pandas. Focus more on the *analysis* of the correlation than on complex machine learning.

- **Example 4: Topic_CheMBL_35_2_Fingerprint_Clustering**
  - **Topic:** Clustering compounds based on their molecular fingerprints.
  - **SQL (Modification):** Keep basic SQL to get `chembl_id`, `smiles` and basic activity.
  - **Python (Modification):**
    - Calculate fingerprints (as in the original code).
    - Use scikit-learn's clustering algorithms (e.g., KMeans, DBSCAN) to cluster the compounds based on their fingerprints.
    - Visualize the clusters using dimensionality reduction techniques like PCA or t-SNE.

```
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Assuming you have X (fingerprint data) from the original code

# Clustering
kmeans = KMeans(n_clusters=5, random_state=42)  # Adjust n_clusters
df['cluster'] = kmeans.fit_predict(X)

# PCA for Visualization
pca = PCA(n_components=2)
pca_result = pca.fit_transform(X)
df['pca_1'] = pca_result[:, 0]
df['pca_2'] = pca_result[:, 1]

# Visualization
plt.figure(figsize=(8, 6))
for cluster in df['cluster'].unique():
    subset = df[df['cluster'] == cluster]
    plt.scatter(subset['pca_1'], subset['pca_2'], label=f'Cluster
{cluster}')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.title('Compound Clustering')
plt.legend()
plt.show()
```

- **Example 5: Topic_CheMBL_35_2_Activity_Threshold_Classification**
  - **Topic:** Classifying compounds as active or inactive based on an activity threshold.
  - **SQL (Modification):** Keep the basic SQL query.
  - **Python (Modification):**
    - Define an activity threshold (e.g., pChEMBL value > 6 is "active").
    - Create a binary classification target variable based on this threshold.
    - Train a classification model (e.g., Logistic Regression, Random Forest) to predict activity class.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
```

```
# Activity threshold
activity_threshold = 6.0

# Create binary target variable
df['active'] = (df['pchembl_value'] > activity_threshold).astype(int)  # 1
for active, 0 for inactive
y = df['active'].values

# Train/Test Split (as in the original code)

# Train a Logistic Regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print(classification_report(y_test, y_pred))
```

Important Considerations and Next Steps:

- **SMILES Retrieval:** The most important next step is to implement the SMILES retrieval. You'll need to either use the ChEMBL API or a more complex SQL query to fetch the SMILES strings based on the `chembl_id`. This is essential for the RDKit descriptor calculation to work correctly.
- **Feature Selection:** The 1024-bit Morgan fingerprints generate a large number of features. Consider using feature selection techniques (e.g., SelectKBest, PCA) to reduce the dimensionality of the feature space and improve model performance.
- **Model Tuning:** Experiment with different machine learning models and tune their hyperparameters using techniques like cross-validation and grid search.
- **Data Exploration:** Perform more in-depth exploratory data analysis (EDA) to understand the distribution of your data, identify potential outliers, and gain insights that can inform your model building process.
- **Error Handling:** Implement robust error handling to gracefully handle unexpected issues during data processing and model training.
- **AIMLops Structure:** Review the overall AIMLops template. Pay close attention to proper version control (e.g., using Git), experiment tracking, and model deployment strategies. Your notebook should be part of a well-defined pipeline.

This comprehensive guide provides a strong foundation for your Chembl 35 data analysis project. Remember to replace the dummy SMILES with actual data retrieval code and to carefully consider the specific goals of your project when choosing models and evaluation metrics. Good luck!