PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: Drug repurposing candidates from ChEMBL

Okay, I understand. You're working with ChEMBL data using RDKit and want to analyze it. You're following an AIMLops structure and need help with your SQL queries, Python code, and some specific errors. Let's break this down step-by-step.

1. Analysis of the Analysis Model

The goal of analyzing ChEMBL data with RDKit is typically to extract information about molecules, their properties, and their interactions with biological targets. This information can then be used for:

- **Virtual Screening:** Predicting the activity of compounds against a target before actually testing them in the lab.
- **Structure-Activity Relationship (SAR) Analysis:** Identifying which parts of a molecule are important for its activity.
- **Drug Repurposing:** Finding new uses for existing drugs.
- **Lead Optimization:** Improving the properties of a promising drug candidate.

Your project uses ChEMBL 35, which is a relatively large dataset. Limiting your initial queries to 100 rows is a smart move to speed up development and testing.

The typical workflow often involves:

1. **Data Extraction:** Using SQL to pull relevant data from the ChEMBL database (e.g., compound structures, activity data).
2. **Data Cleaning/Preprocessing:** Handling missing data, standardizing compound representations (e.g., using SMILES strings), and filtering out unwanted data.
3. **Feature Engineering:** Generating molecular descriptors using RDKit. These descriptors represent various aspects of the molecule's structure, such as its size, shape, and electronic properties.
4. **Model Building:** Training a machine learning model (e.g., regression, classification) to predict activity based on the molecular descriptors.
5. **Model Evaluation:** Assessing the performance of the model using appropriate metrics (e.g., R-squared, RMSE, AUC).
6. **Interpretation/Visualization:** Analyzing the model to understand which descriptors are most important for activity and visualizing the results.

You have some error need to resolve: * "ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '[1]+$'" * "old scikit-learn version does not support parameters squared=False in the mean_squared_error function"

English and Vietnamese Explanations:

English:

- **Understanding the Errors:**
    - **SQL Error:** `ERROR: operator does not exist: numeric ~ unknown` This error indicates that you're trying to use the regular expression matching operator `~` (similar to `LIKE` but for regular expressions) on a numeric column (`act.standard_value`).

[1] 0-9.

PostgreSQL doesn't directly support regular expression matching on numeric data types. You need to cast the column to a text type.
    - **Python Error:** `old scikit-learn version does not support parameters squared=False` This means the version of scikit-learn you are using for MSE calculation is outdated. You need to upgrade scikit-learn or adjust your code.
- **How to Fix:**
    - **SQL Fix:** Use `CAST(act.standard_value AS TEXT) ~ '^[0-9\.]+$'` to convert the `standard_value` column to text before applying the regular expression.
    - **Python Fix:** Update your scikit-learn with `pip install -U scikit-learn`. If upgrading is not an option, remove the argument `squared=False` from `mean_squared_error` and take the square root of the result manually to obtain RMSE (Root Mean Squared Error).

Vietnamese:

- **Hiểu các lỗi:**
    - **Lỗi SQL:** `ERROR: operator does not exist: numeric ~ unknown` Lỗi này chỉ ra rằng bạn đang cố gắng sử dụng toán tử so khớp biểu thức chính quy ~ (tương tự như `LIKE` nhưng dành cho biểu thức chính quy) trên một cột kiểu số (`act.standard_value`). PostgreSQL không hỗ trợ trực tiếp việc so khớp biểu thức chính quy trên kiểu dữ liệu số. Bạn cần chuyển đổi cột đó sang kiểu text.
    - **Lỗi Python:** `old scikit-learn version does not support parameters squared=False` Điều này có nghĩa là phiên bản scikit-learn bạn đang sử dụng để tính MSE đã cũ. Bạn cần nâng cấp scikit-learn hoặc điều chỉnh code của mình.
- **Cách sửa:**
    - **Sửa lỗi SQL:** Sử dụng `CAST(act.standard_value AS TEXT) ~ '^[0-9\.]+$'` để chuyển đổi cột `standard_value` thành text trước khi áp dụng biểu thức chính quy.
    - **Sửa lỗi Python:** Cập nhật scikit-learn của bạn bằng `pip install -U scikit-learn`. Nếu việc nâng cấp không khả thi, hãy xóa đối số `squared=False` khỏi `mean_squared_error` và tự tính căn bậc hai của kết quả để có được RMSE (Root Mean Squared Error).

2. SQL and Python Code

Here's the code with fixes, limited to 100 rows, and structured according to your requirements.

SQL Code (saved as `../data/Topic_CheMBL_35_46.csv`):

```sql
-- Topic_CheMBL_35_46.sql
SELECT
    mol.molregno,
    mol.smiles,
    act.standard_value,
    act.standard_units,
    act.activity_comment,
    act.standard_type,
    target.target_type,
    target.organism,
    cmpd.compound_name
FROM
    compound_structures mol
JOIN
    activities act ON mol.molregno = act.molregno
JOIN
    target_dictionary target ON act.tid = target.tid
LEFT JOIN
    compound_properties cmpd ON mol.molregno = cmpd.molregno
WHERE act.standard_type = 'IC50'
```

```
  AND act.standard_units = 'nM'
  AND CAST(act.standard_value AS TEXT) ~ '^[0-9\.]+$'  -- Fixed the numeric ~ unknown
error
  AND act.standard_value IS NOT NULL
  AND mol.smiles IS NOT NULL
LIMIT 100;
```

Python Code (saved as `notebook/Topic_CheMBL_35_46_1_data_prep.ipynb`):

```
# Topic_CheMBL_35_46_1_data_prep.ipynb

import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import AllChem
import numpy as np

# Define base path
base_path = "../data" # Assuming data folder is one level up from the notebook folder

# Input file
input_file = os.path.join(base_path, "Topic_CheMBL_35_46.csv")

# Read the data from CSV
try:
    df = pd.read_csv(input_file)
    print(f"Data loaded successfully from {input_file}")
except FileNotFoundError:
    print(f"Error: File not found at {input_file}")
    df = None  # or exit()
except Exception as e:
    print(f"An error occurred while reading the CSV: {e}")
    df = None # or exit()

if df is not None:
    # Basic data cleaning (remove rows with missing SMILES or standard_value)
    df = df.dropna(subset=['smiles', 'standard_value'])

    # Convert standard_value to numeric (after cleaning)
    df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')  #
Handle potential conversion errors
    df = df.dropna(subset=['standard_value']) # drop rows with NaN in standard_value
after coercion to numeric.

    # Convert IC50 to pIC50 (optional, but common) - more stable for modelling
    df = df[df['standard_value'] > 0] # Remove values <= 0 before log transform
    df['pIC50'] = -np.log10(df['standard_value'] * 1e-9) # Convert nM to M and take
-log10
    print(df.head())
```

Python Code (saved as `notebook/Topic_CheMBL_35_46_2_model_building.ipynb`):

```
# Topic_CheMBL_35_46_2_model_building.ipynb

import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import AllChem
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

# Define base path
base_path = "../data" # Assuming data folder is one level up from the notebook folder

# Input file (same as previous notebook)
input_file = os.path.join(base_path, "Topic_CheMBL_35_46.csv")

# Read the data from CSV
try:
    df = pd.read_csv(input_file)
    print(f"Data loaded successfully from {input_file}")
except FileNotFoundError:
    print(f"Error: File not found at {input_file}")
    df = None  # or exit()
except Exception as e:
    print(f"An error occurred while reading the CSV: {e}")
    df = None # or exit()

if df is not None:
    # Basic data cleaning (remove rows with missing SMILES or standard_value)
    df = df.dropna(subset=['smiles', 'standard_value'])

    # Convert standard_value to numeric (after cleaning)
    df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')  #
Handle potential conversion errors
    df = df.dropna(subset=['standard_value']) # drop rows with NaN in standard_value
after coercion to numeric.

    # Convert IC50 to pIC50 (optional, but common) - more stable for modelling
    df = df[df['standard_value'] > 0] # Remove values <= 0 before log transform
    df['pIC50'] = -np.log10(df['standard_value'] * 1e-9) # Convert nM to M and take
-log10

    # Generate Morgan fingerprints (ECFP4) as molecular descriptors
    def generate_morgan_fingerprint(smiles, radius=2, nBits=2048):
        try:
            mol = Chem.MolFromSmiles(smiles)
            if mol:
                fp = AllChem.GetMorganFingerprintAsBitVect(mol, radius, nBits=nBits)
                return np.array(fp)
            else:
                return None
        except:
            return None

    df['fingerprint'] = df['smiles'].apply(generate_morgan_fingerprint)
    df = df.dropna(subset=['fingerprint'])  # Remove rows where fingerprint generation
failed

    # Prepare data for modeling
    X = np.stack(df['fingerprint'])  # Convert fingerprints to a NumPy array
    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)
```

```
# Standardize the features (important for linear models)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train a linear regression model
model = LinearRegression()
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 the evaluation metrics
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")
```

Explanation:

- **SQL:** The SQL query selects key information (SMILES, activity, target, etc.) from the ChEMBL database. It filters for IC50 values in nM and includes the fix for the numeric column comparison. It limits the result to 100 rows for quicker execution.
- **Python (Data Preparation):** This notebook reads the CSV file generated by the SQL query. It cleans the data by removing rows with missing values. The `standard_value` is converted to numeric and then to pIC50 (a more common and stable representation for activity modeling).
- **Python (Model Building):** This notebook builds a simple linear regression model to predict pIC50 values based on Morgan fingerprints (ECFP4) generated from the SMILES strings. It splits the data into training and testing sets, standardizes the features, trains the model, and evaluates its performance using MSE and R-squared. It assumes you have `scikit-learn` and `RDKit` installed.

3. Five Example Scenarios

Here are five examples of how you can use this code and adapt it to different scenarios:

1. **Target Specific Analysis:**
    - **Scenario:** You want to focus on compounds active against a specific target, say, 'CHEMBL205' (a specific kinase).
    - **Modification:** Modify the SQL query to include `AND target.tid = 'CHEMBL205'` in the `WHERE` clause. Rerun the SQL query and the Python notebooks. This will restrict your analysis to only compounds that target this specific target.
2. **Different Activity Type:**
    - **Scenario:** You are interested in Ki (inhibition constant) instead of IC50 values.
    - **Modification:** Modify the SQL query to change `act.standard_type = 'IC50'` to `act.standard_type = 'Ki'`. You might also need to adjust the units filter (`act.standard_units`) accordingly. Re-run the SQL and Python notebooks.
3. **Exploring Different Molecular Descriptors:**
    - **Scenario:** You want to use different molecular descriptors, such as physicochemical properties calculated by RDKit (e.g., molecular weight, logP).
    - **Modification:** In the `Topic_CheMBL_35_46_2_model_building.ipynb` notebook, remove the Morgan fingerprint generation code and replace it with code to calculate other descriptors. Here's an example:

```
from rdkit.Chem import Descriptors

def calculate_descriptors(smiles):
    try:
        mol = Chem.MolFromSmiles(smiles)
        if mol:
            mw = Descriptors.MolWt(mol)
            logp = Descriptors.MolLogP(mol)
            # Add more descriptors as needed
            return pd.Series([mw, logp]) # Return as a Pandas Series
        else:
            return pd.Series([None, None]) # Return as Pandas Series of None values
    except:
        return pd.Series([None, None]) # Return as Pandas Series of None values


df[['MW', 'LogP']] = df['smiles'].apply(calculate_descriptors) # Apply and assign the result directly
df = df.dropna(subset=['MW', 'LogP']) # remove rows with missing values.

# Prepare data for modeling (using MW and LogP)
X = df[['MW', 'LogP']].values  # Select the MW and LogP columns
```

You'll then need to adjust the feature scaling and model training parts of the notebook accordingly.

4. **Using a Different Machine Learning Model:**
    - **Scenario:** You want to try a different machine learning model, such as a Random Forest Regressor.
    - **Modification:** In the `Topic_CheMBL_35_46_2_model_building.ipynb` notebook, replace the `LinearRegression` model with `RandomForestRegressor`.

```
from sklearn.ensemble import RandomForestRegressor

# Train a Random Forest Regression model
model = RandomForestRegressor(n_estimators=100, random_state=42) # Adjust parameters as needed
model.fit(X_train, y_train)
```

You might need to adjust the model's hyperparameters for optimal performance.

5. **Analyzing Activity Comments:**
    - **Scenario:** You want to investigate why some activities have comments (e.g., "inactive," "weak activity").
    - **Modification:** In the SQL query, remove the `LIMIT 100` clause (or increase it significantly) to get a larger dataset. In a new Python notebook, load the data and analyze the `activity_comment` column. You can use techniques like:
        - **Value counts:** `df['activity_comment'].value_counts()` to see the frequency of different comments.
        - **Filtering:** `df[df['activity_comment'] == 'inactive']` to select only inactive compounds and analyze their properties.
        - **Text analysis:** Using natural language processing (NLP) techniques to extract information from the comments.

Important Considerations:

- **RDKit Installation:** Ensure you have RDKit installed in your Python environment (`conda install -c conda-forge rdkit`).

- **File Paths:** Double-check that the file paths in your Python code are correct. The `os.path.join` function is a good way to ensure this.
- **Data Exploration:** Before building models, always spend time exploring your data. Look at distributions, correlations, and potential biases.
- **Feature Scaling:** Scaling your features (e.g., using `StandardScaler`) is often crucial for the performance of machine learning models, especially linear models.
- **Model Evaluation:** Use appropriate evaluation metrics for your task. R-squared is useful for regression, but metrics like AUC or F1-score are more appropriate for classification.
- **Version Control:** Use Git for version control to track your changes and collaborate effectively.

This comprehensive breakdown should get you started. Remember to adapt the code and analysis to your specific research questions and data. Good luck!