PharmApp Suite

🧠 AI for Drug Discovery and Development 🧪

Copyright 2025 | RnD Pharma Plus | www.nghiencuuthuoc.com | Zalo: +84888999311 |

🧩 Topic: Trends in target class approval

Okay, I understand. You're working with ChEMBL data using RDKit, following an AIMLops structure, and facing some issues. Let's break down the analysis, address the errors, and provide you with SQL and Python code snippets.

1. Analysis Model & Overview:

The goal is to extract, process, and analyze chemical and bioactivity data from ChEMBL 35 using SQL and Python (RDKit, scikit-learn). Here's a general analysis model:

- **Data Extraction (SQL):** Fetch relevant data from the ChEMBL database tables using SQL queries. This includes compound information (SMILES strings), bioactivity data (IC50, Ki, etc.), and target information. The SQL queries should filter for a small subset of data (100 rows) to avoid overwhelming your system.
- **Data Preparation (Python - RDKit):** Use RDKit to process the SMILES strings into molecular objects. Calculate molecular descriptors (e.g., molecular weight, LogP, number of hydrogen bond donors/acceptors).
- **Data Cleaning and Filtering (Python):** Remove or correct any invalid or incomplete data points. Ensure that the bioactivity data is in a consistent unit and scale. Filter out inactive compounds or those with unreliable measurements.
- **Exploratory Data Analysis (EDA) (Python):** Visualize and summarize the data. Look for trends, patterns, and correlations between molecular descriptors and bioactivity. Histograms, scatter plots, and box plots are useful for this step.
- **Modeling (Python - scikit-learn):** Build statistical or machine learning models to predict bioactivity based on molecular descriptors. Common models include linear regression, random forests, and support vector machines.
- **Model Evaluation (Python - scikit-learn):** Evaluate the performance of the models using appropriate metrics (e.g., R-squared, RMSE, AUC).

Addressing Errors:

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

  This error indicates that you're trying to use a regular expression operator (~) with a numeric column (`act.standard_value`). PostgreSQL doesn't directly support regular expression matching on numeric types. The issue arises from your attempt to filter `standard_value` to only include numeric values using a regex. The best approach is to cast the value to TEXT first.

- **Error b: `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. Either upgrade scikit-learn, or remove the `squared=False` parameter from your `mean_squared_error` call.

2. SQL and Python Code Examples (with explanations):

Here's the code, split into SQL and Python sections.

SQL (pgAdmin - `../data/chembl_35_50_data.csv`)

```sql
-- Topic_CheMBL_35_50 - Retrieve Compound Data and Activity
-- Get 100 rows of data

SELECT
    md.molregno,
    cs.canonical_smiles,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    act.assay_id,
    td.pref_name AS target_name
FROM
    molecule_dictionary md
JOIN
    compound_structures cs ON md.molregno = cs.molregno
JOIN
    activities act ON md.molregno = act.molregno
JOIN
    target_dictionary td ON act.tid = td.tid
WHERE
    act.standard_type = 'IC50' -- Focus on IC50 values
    AND act.standard_units = 'nM'  -- Focus on nM units
    AND act.standard_value IS NOT NULL
    AND cs.canonical_smiles IS NOT NULL
    AND act.standard_value::TEXT ~ '^[0-9\.]+$' --Cast to TEXT and keep only numeric
values.
LIMIT 100;

--Save the results to chembl_35_50_data.csv
```

Explanation:

- The SQL query joins tables: `molecule_dictionary`, `compound_structures`, `activities`, and `target_dictionary` to retrieve relevant information.
- `act.standard_type = 'IC50'` and `act.standard_units = 'nM'` filter the data to focus on IC50 values measured in nanomolars.
- `act.standard_value IS NOT NULL` and `cs.canonical_smiles IS NOT NULL` exclude rows with missing values.
- `LIMIT 100` restricts the output to 100 rows.
- `act.standard_value::TEXT ~ '^[0-9\.]+$'` now includes `::TEXT` to explicitly cast `standard_value` to a string before applying the regular expression.

Python (Jupyter Notebook - `notebook/Topic_CheMBL_35_50_1_Data_Processing.ipynb`)

```python
import pandas as pd
import numpy as np
from rdkit import Chem
from rdkit.Chem import Descriptors
from rdkit.Chem import Lipinski
import os
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns

# Define base path
base_path = "../data"  # Assuming your notebook is in the 'notebook' directory
data_file = "chembl_35_50_data.csv"
```

```
data_path = os.path.join(base_path, data_file)

# Load the data
try:
    df = pd.read_csv(data_path)
except FileNotFoundError:
    print(f"Error: File not found at {data_path}.  Make sure you've run the SQL query
and saved the CSV.")
    exit()

print(f"Data loaded successfully. Shape: {df.shape}")
print(df.head())

# Data Cleaning and Preprocessing
df.dropna(subset=['canonical_smiles', 'standard_value'], inplace=True)
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce') #Convert
to numeric and handle errors.
df.dropna(subset=['standard_value'], inplace=True)
df = df[df['standard_value'] > 0] #Remove zero or negative values
df = df[df['standard_value'] < 100000] #Remove values > 100 uM


# RDKit Molecular Descriptors Calculation
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is not None:
        mw = Descriptors.MolWt(mol)
        logp = Descriptors.MolLogP(mol)
        hbd = Lipinski.NumHDonors(mol)
        hba = Lipinski.NumHAcceptors(mol)
        return mw, logp, hbd, hba
    else:
        return None, None, None, None

df[['mw', 'logp', 'hbd', 'hba']] = df['canonical_smiles'].apply(lambda x:
pd.Series(calculate_descriptors(x)))
df.dropna(inplace=True) #remove rows where descriptor calculation failed

# Data Transformation (e.g., log transformation of IC50 values)
df['pIC50'] = -np.log10(df['standard_value'] / 1e9)  # Convert IC50 (nM) to pIC50

# Feature Selection (example: using molecular weight and LogP)
X = df[['mw', 'logp', 'hbd', 'hba']]
y = df['pIC50']

# Data Splitting
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 Evaluation
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred) #Remove squared=False
r2 = r2_score(y_test, y_pred)

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

```python
print(f"R-squared: {r2}")

# Visualization (example: Scatter plot of predicted vs. actual pIC50)
plt.scatter(y_test, y_pred)
plt.xlabel("Actual pIC50")
plt.ylabel("Predicted pIC50")
plt.title("Predicted vs. Actual pIC50")
plt.show()
```

Explanation:

1. **Import Libraries:** Imports necessary libraries like pandas, RDKit, scikit-learn.
2. **Define Paths:** Constructs file paths using `os.path.join` for portability.
3. **Load Data:** Reads the CSV file into a pandas DataFrame. Includes error handling for the file not being found.
4. **Data Cleaning:** Handles missing values and ensures `standard_value` is numeric. Filters out values to be within a reasonable range.
5. **Descriptor Calculation:** Calculates molecular descriptors using RDKit. Includes a function `calculate_descriptors` that handles potential errors in molecule parsing.
6. **Data Transformation:** Transforms IC50 values to pIC50 (a more common scale).
7. **Feature Selection:** Selects molecular weight (mw) and LogP as features.
8. **Data Splitting:** Splits the data into training and testing sets.
9. **Model Training:** Trains a linear regression model.
10. **Model Evaluation:** Evaluates the model using Mean Squared Error (MSE) and R-squared (R2).
11. **Visualization:** Creates a scatter plot to visualize the model's performance.

3. Five Examples:

Here are five examples demonstrating different aspects of this workflow. These can be incorporated into your notebook.

1. **Example 1: Filtering by Target Name:** Modify the SQL query to retrieve data only for a specific target protein. sql `-- SQL: Filter by Target Name SELECT ... FROM ... WHERE act.standard_type = 'IC50' AND act.standard_units = 'nM' AND act.standard_value IS NOT NULL AND cs.canonical_smiles IS NOT NULL AND td.pref_name = 'CHEMBL205' -- Example: Estrogen receptor alpha LIMIT 100;`

2. **Example 2: Calculating More Descriptors:** Add more molecular descriptors using RDKit.

   ```python
   # Python: Add more descriptors
   def calculate_descriptors(smiles):
       mol = Chem.MolFromSmiles(smiles)
       if mol is not None:
           mw = Descriptors.MolWt(mol)
           logp = Descriptors.MolLogP(mol)
           hbd = Lipinski.NumHDonors(mol)
           hba = Lipinski.NumHAcceptors(mol)
           tpsa = Descriptors.TPSA(mol)  # Topological Polar Surface Area
           rotatable_bonds = Lipinski.NumRotatableBonds(mol)
           return mw, logp, hbd, hba, tpsa, rotatable_bonds
       else:
           return None, None, None, None, None, None

   df[['mw', 'logp', 'hbd', 'hba', 'tpsa', 'rotatable_bonds']] =
   df['canonical_smiles'].apply(lambda x: pd.Series(calculate_descriptors(x)))
   df.dropna(inplace=True)
   ```

3. **Example 3: Using a Different Model:** Use a Random Forest Regressor instead of Linear Regression.

   ```
   # Python: Random Forest Model
   from sklearn.ensemble import RandomForestRegressor

   model = RandomForestRegressor(n_estimators=100, random_state=42)  # Example
   parameters
   model.fit(X_train, y_train)

   y_pred = model.predict(X_test)
   mse = mean_squared_error(y_test, y_pred)
   r2 = r2_score(y_test, y_pred)

   print(f"Random Forest MSE: {mse}")
   print(f"Random Forest R-squared: {r2}")
   ```

4. **Example 4: Log Transformation of IC50 and Visualization:** Visualize the distribution of IC50 values before and after log transformation.

   ```
   # Python: IC50 Distribution
   plt.figure(figsize=(12, 5))
   plt.subplot(1, 2, 1)
   sns.histplot(df['standard_value'], kde=True)
   plt.title("Distribution of IC50 (nM)")

   plt.subplot(1, 2, 2)
   sns.histplot(df['pIC50'], kde=True)
   plt.title("Distribution of pIC50")

   plt.show()
   ```

5. **Example 5: Saving the Model:** Save the trained model to a file.

   ```
   # Python: Save the model
   import pickle

   model_filename = os.path.join(base_path, "chembl_model.pkl")
   pickle.dump(model, open(model_filename, 'wb'))
   print(f"Model saved to {model_filename}")

   # To load the model later:
   # loaded_model = pickle.load(open(model_filename, 'rb'))
   ```

Important Considerations:

- **Error Handling:** Include robust error handling in your Python code (e.g., `try...except` blocks) to handle potential issues like invalid SMILES strings or missing data.
- **Data Validation:** Validate your data at each stage to ensure data quality.
- **Feature Engineering:** Experiment with different molecular descriptors and feature engineering techniques to improve model performance.
- **Model Selection:** Evaluate different machine learning models and choose the one that performs best for your specific task.
- **Hyperparameter Tuning:** Optimize the hyperparameters of your chosen model using techniques like cross-validation.
- **Explainability:** Consider using techniques to make your models more explainable, such as SHAP values or LIME.

Analysis in Vietnamese (Phân tích bằng tiếng Việt):

1. **Mô hình phân tích:** Mục tiêu là trích xuất, xử lý và phân tích dữ liệu hóa học và hoạt tính sinh học từ ChEMBL 35. Chúng ta sẽ sử dụng SQL để lấy dữ liệu, RDKit để xử lý cấu trúc phân tử, và scikit-learn để xây dựng mô hình dự đoán hoạt tính.
2. **Xử lý lỗi:**
    - Lỗi `ERROR: operator does not exist: numeric ~ unknown`: Lỗi này xảy ra khi bạn cố gắng sử dụng toán tử so khớp biểu thức chính quy (~) trên cột số (`act.standard_value`). Để khắc phục, hãy chuyển đổi giá trị số thành chuỗi (TEXT) trước khi so khớp biểu thức chính quy.
    - Lỗi `old scikit-learn version does not support parameters squared=False`: Bạn đang sử dụng phiên bản scikit-learn cũ. Hãy nâng cấp phiên bản hoặc loại bỏ tham số `squared=False` khỏi hàm `mean_squared_error`.
3. **Code:** Các đoạn code SQL và Python được cung cấp ở trên, kèm theo giải thích chi tiết.
4. **Ví dụ:** Năm ví dụ minh họa các khía cạnh khác nhau của quy trình làm việc này, giúp bạn hiểu rõ hơn cách áp dụng các kỹ thuật khác nhau.

By following these steps and addressing the errors, you should be able to successfully run your Chembl 35 data analysis project and gain valuable insights for drug research and development. Remember to adapt the code to your specific needs and experiment with different parameters and models. Good luck!