PharmApp Suite

🧠 AI for Drug Discovery and Development 🍼

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

🧩 Topic: XGBoost optimization for IC50 prediction

Okay, I understand. You're working with ChEMBL 35 data, using RDKit within a AIMLops framework, and encountering issues with your PostgreSQL queries and scikit-learn version. You need guidance on structuring your analysis, SQL queries, Python code, and handling the identified errors, all while staying within a limited dataset size (100 rows) for efficiency. Let's break this down systematically.

1. Analysis Model & Approach

Your overall goal is to extract and analyze drug research and development data from ChEMBL 35. A common approach is to:

- **Data Extraction and Filtering:** Use SQL to extract relevant data from ChEMBL, focusing on specific tables like `activities`, `molecule_dictionary`, `compound_structures`, etc. You'll need to filter this data based on criteria important for your research question (e.g., target protein, assay type, activity values). Limiting the data to 100 rows *after* filtering ensures you have a representative sample without overwhelming your system.
- **Data Cleaning and Transformation:** Use Python and RDKit to clean and transform the extracted data. This involves handling missing values, converting SMILES strings to RDKit Mol objects, calculating molecular descriptors, and transforming activity values (e.g., IC50, Ki) into pChEMBL.
- **Exploratory Data Analysis (EDA):** Perform EDA to understand the data distribution, identify potential correlations between molecular descriptors and activity, and visualize key features.
- **Modeling (Optional):** Based on your research question, you might build predictive models to relate molecular properties to activity.
- **Interpretation and Reporting:** Summarize your findings and draw conclusions based on the analysis.

2. Directory Structure and File Naming (AIMLops Compliance)

You mentioned following the AIMLops standard. This typically means organizing your project with a structure like:

```
project_root/
├── data/            # SQL query results (*.csv)
├── notebooks/       # Jupyter notebooks ("Topic_CheMBL_35_63_1_*.ipynb",
"Topic_CheMBL_35_63_2_*.ipynb", etc.)
├── scripts/         # (Optional) Python scripts for reusable functions
├── models/          # (Optional) Trained models
├── reports/         # (Optional) Analysis reports
├── README.md
```

Make sure you follow this structure consistently. `os.path.join(base_path, ...)` is the correct way to construct file paths within your code, ensuring portability.

3. Addressing the Errors

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

  This error arises because you're trying to use regular expression matching (~) on a numeric column (`act.standard_value`). You need to cast it to text first. Also, the `standard_value`

column in ChEMBL *should* already contain numeric values, so filtering for "numbers and periods" is likely unnecessary and *may lead to losing data*. A more appropriate filter would be to simply check if the value is not null and is within a reasonable range for biological activity.

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

  This is a version incompatibility issue. You have two options:

  1. **Upgrade scikit-learn:** This is the preferred approach. Use `pip install --upgrade scikit-learn` in your environment.
  2. **Remove `squared=False`:** If upgrading isn't possible, remove the `squared=False` argument. The function will then return the Mean Squared Error (MSE) instead of the Root Mean Squared Error (RMSE). You can then take the square root of the result to get the RMSE. This is a workaround, but less desirable.

4. SQL Code Example (PostgreSQL)

```sql
-- Topic_CheMBL_35_63_extract_data.sql
-- Extracts ChEMBL data, limited to 100 rows for testing.

SELECT
    md.chembl_id,
    cs.canonical_smiles,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    act.target_chembl_id
FROM
    activities act
JOIN
    molecule_dictionary md ON act.molregno = md.molregno
JOIN
    compound_structures cs ON md.molregno = cs.molregno
WHERE
    act.standard_type = 'IC50'  -- Example filter:  Focus on IC50 values
    AND act.standard_units = 'nM' -- Example filter: Focus on nM units
    AND act.standard_value IS NOT NULL  -- Ensure standard_value is not null
    AND act.standard_value > 0      -- Exclude zero values
    AND act.standard_value < 10000   -- Example filter:  reasonable range for IC50
LIMIT 100;
```

Important:

- **Save this SQL query as a `.sql` file (e.g., `Topic_CheMBL_35_63_extract_data.sql`).**
- **Run this query in pgAdmin connected to your `chembl_35` database.**
- **Export the result of the query to a CSV file (e.g., `Topic_CheMBL_35_63_data.csv`) and save it in your `data/` directory.** Make sure to include the header row when exporting.

5. Python Code Example (Jupyter Notebook - Topic_CheMBL_35_63_1_data_prep.ipynb)

```python
# Topic_CheMBL_35_63_1_data_prep.ipynb

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

# --- Configuration ---
```

```python
base_path = os.getcwd()  # Get current working directory
data_file = os.path.join(base_path, 'data', 'Topic_CheMBL_35_63_data.csv')
output_file = os.path.join(base_path, 'data', 'Topic_CheMBL_35_63_processed_data.csv')

# --- Load Data ---
try:
    df = pd.read_csv(data_file)
    print(f"Data loaded successfully from: {data_file}")
except FileNotFoundError:
    print(f"Error: File not found at {data_file}.  Make sure you ran the SQL query and
saved the CSV.")
    exit()

# --- Data Cleaning and Transformation ---

# 1. Handle missing SMILES
df = df.dropna(subset=['canonical_smiles'])
print(f"Number of rows after dropping NA SMILES: {len(df)}")

# 2. Convert IC50 to pIC50
def ic50_to_pic50(ic50_nM):
    """Converts IC50 (nM) to pIC50."""
    pIC50 = -np.log10(ic50_nM * 1e-9)  # Convert nM to Molar
    return pIC50

df['pIC50'] = df['standard_value'].apply(ic50_to_pic50)

# 3. RDKit Mol Object and Molecular Descriptors
def calculate_descriptors(smiles):
    """Calculates molecular descriptors using RDKit."""
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None  # Handle invalid SMILES
    descriptors = {}
    descriptors['MW'] = Descriptors.MolWt(mol)
    descriptors['LogP'] = Descriptors.MolLogP(mol)
    # Add more descriptors as needed
    return descriptors

df['descriptors'] = df['canonical_smiles'].apply(calculate_descriptors)

# Handle rows where descriptor calculation failed (invalid SMILES)
df = df[df['descriptors'].notna()]

# Separate descriptors into individual columns
df = pd.concat([df.drop(['descriptors'], axis=1), df['descriptors'].apply(pd.Series)],
axis=1)

# --- Save Processed Data ---
df.to_csv(output_file, index=False)
print(f"Processed data saved to: {output_file}")
print(df.head())
```

Python Code Example (Jupyter Notebook - Topic_CheMBL_35_63_2_eda.ipynb)

```python
# Topic_CheMBL_35_63_2_eda.ipynb

import os
import pandas as pd
import matplotlib.pyplot as plt
```

```
import seaborn as sns

# --- Configuration ---
base_path = os.getcwd()
processed_data_file = os.path.join(base_path, 'data',
'Topic_CheMBL_35_63_processed_data.csv')

# --- Load Processed Data ---
try:
    df = pd.read_csv(processed_data_file)
    print(f"Processed data loaded successfully from: {processed_data_file}")
except FileNotFoundError:
    print(f"Error: File not found at {processed_data_file}.  Make sure you ran the
data prep notebook.")
    exit()

# --- Exploratory Data Analysis ---

# 1. Distribution of pIC50
plt.figure(figsize=(8, 6))
sns.histplot(df['pIC50'], kde=True)
plt.title('Distribution of pIC50 Values')
plt.xlabel('pIC50')
plt.ylabel('Frequency')
plt.show()

# 2. Scatter plot of Molecular Weight vs. pIC50
plt.figure(figsize=(8, 6))
sns.scatterplot(x='MW', y='pIC50', data=df)
plt.title('Molecular Weight vs. pIC50')
plt.xlabel('Molecular Weight (MW)')
plt.ylabel('pIC50')
plt.show()

# 3. Box plot of pIC50 by Target
plt.figure(figsize=(12, 6))
sns.boxplot(x='target_chembl_id', y='pIC50', data=df)
plt.title('pIC50 Distribution by Target')
plt.xlabel('Target ChEMBL ID')
plt.ylabel('pIC50')
plt.xticks(rotation=90)  # Rotate x-axis labels for readability
plt.show()

# 4. Correlation Matrix (Numerical Features)
numerical_cols = ['standard_value', 'pIC50', 'MW', 'LogP']  # Include numerical
columns
corr_matrix = df[numerical_cols].corr()

plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()

# 5. Value counts of standard_type and standard_units
print("Value counts for standard_type:")
print(df['standard_type'].value_counts())

print("\nValue counts for standard_units:")
print(df['standard_units'].value_counts())
```

6. Examples

Here are five concrete examples of how you might use this framework for drug discovery research:

1. **Target-Based Analysis:** Focus on a specific protein target (e.g., a kinase involved in cancer). Filter the SQL query by `act.target_chembl_id` to extract data for compounds tested against that target. Analyze the relationship between molecular descriptors and activity for that target.
2. **Lead Optimization:** Identify a set of compounds with moderate activity and explore how changes in their structure (based on SMILES strings) affect their potency. Calculate descriptors for analogs and build a model to predict activity changes.
3. **Structure-Activity Relationship (SAR) Analysis:** Visualize the activity of compounds with similar structures using RDKit's substructure highlighting capabilities. This can help identify key structural features that contribute to activity.
4. **Virtual Screening Validation:** Use the ChEMBL data to validate the results of a virtual screening campaign. Compare the predicted activity of compounds identified in the virtual screen with their actual activity values in ChEMBL.
5. **Cross-Target Activity:** Investigate whether compounds active against one target also show activity against other targets. This could reveal potential off-target effects or polypharmacology.

Important Considerations:

- **Error Handling:** Include `try...except` blocks in your Python code to handle potential errors during data loading, SMILES parsing, and descriptor calculation.
- **Data Validation:** Always validate the data you extract from ChEMBL. Check for inconsistencies, outliers, and potential errors.
- **Documentation:** Document your code and analysis thoroughly. Explain your assumptions, methods, and findings clearly.
- **Version Control:** Use Git to track your code changes and collaborate effectively.

In Summary:

This provides a comprehensive guide to your ChEMBL 35 data analysis project within an AIMLops framework. You have SQL code for data extraction, Python code for data cleaning, transformation, and EDA, and clear instructions on how to address the errors you've encountered. Good luck!