PharmApp Suite

🧠 AI for Drug Discovery and Development 🍼

🧩 Topic: Target-ligand network graph using NetworkX

Okay, I understand. You're working with ChEMBL 35 data using RDKit in a Jupyter Notebook, following an AIMLops template, and facing specific errors while querying and processing the data. Let's break down the problem and provide solutions.

Analysis of the Analysis Model

The project aims to leverage the ChEMBL 35 database (containing information on bioactive molecules, targets, and bioassays) to support drug research and development. You're using RDKit for cheminformatics tasks, likely involving:

- **Data Retrieval (SQL):** Fetching relevant data from the ChEMBL 35 database, such as activity data, compound structures (SMILES strings), and target information. Filtering is crucial to manage the data size.
- **Data Preprocessing (Python/RDKit):** Cleaning, standardizing, and converting chemical structures (SMILES to RDKit Mol objects).
- **Feature Generation (Python/RDKit):** Calculating molecular descriptors (e.g., physicochemical properties, topological indices) from the chemical structures.
- **Data Analysis and Modeling (Python/Scikit-learn):** Performing statistical analysis, building predictive models (e.g., regression models to predict activity, classification models to predict activity type), and visualizing the data.

Addressing the 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 (~) on a numeric column (`act.standard_value`). PostgreSQL's ~ operator is for string matching. The goal here is likely to filter for `standard_value` that *look like* numbers. We can achieve this by explicitly casting the `standard_value` to text or using a different approach.

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

  This error is straightforward. Your scikit-learn version is outdated. Either upgrade scikit-learn to a version that supports the `squared=False` parameter (which returns the root mean squared error instead of the mean squared error) or remove/omit the `squared=False` argument to default to MSE, then take the square root manually if needed. Upgrading is *highly recommended*.

File Structure (AIMLops Compliance)

Following the AIMLops template, a typical structure would look like this:

```
Topic_CheMBL_35_76/
├── data/          # CSV files extracted from ChEMBL
├── notebooks/     # Jupyter Notebooks
│   ├── Topic_CheMBL_35_76_1_data_extraction.ipynb
│   ├── Topic_CheMBL_35_76_2_data_preprocessing.ipynb
│   ├── Topic_CheMBL_35_76_3_feature_engineering.ipynb
│   ├── Topic_CheMBL_35_76_4_model_building.ipynb
```

```
│   └── Topic_CheMBL_35_76_5_evaluation.ipynb
├── src/            # Python modules (if you modularize your code)
├── models/         # Saved model files
└── README.md       # Project description
```

English and Chinese Guidance

Here's the code, with explanations in both English and Chinese:

1. SQL Code (English & Chinese)

This SQL code will extract data from ChEMBL 35, focusing on activity data. It limits the result to 100 rows. The `WHERE` clause is modified to avoid the "numeric ~ unknown" error. I'm assuming you're interested in IC50 values for a specific target of interest (you'll need to replace `'target_id'` with an actual target ID from ChEMBL).

```sql
-- English
-- Extracting activity data for a specific target from ChEMBL 35
-- Limiting to 100 rows for demonstration purposes.
-- Avoiding the numeric ~ operator error.
SELECT
    act.molregno,
    act.standard_type,
    act.standard_relation,
    act.standard_value,
    act.standard_units,
    cmpd.smiles
FROM activities act
JOIN assays ass ON act.assay_id = ass.assay_id
JOIN target_dictionary td ON ass.tid = td.tid
JOIN molecule_dictionary md ON act.molregno = md.molregno
JOIN compound_structures cmpd ON md.molregno = cmpd.molregno
WHERE td.chembl_id = 'CHEMBL206'  -- Replace with your target's ChEMBL ID
  AND act.standard_type = 'IC50' --Filtering IC50 values
  AND act.standard_value IS NOT NULL -- Exclude null values
  AND act.standard_value != '' -- Exclude empty strings
  AND act.standard_value::TEXT ~ '^[0-9\.]+$' -- Ensure that value contains numeric
LIMIT 100;


-- Chinese
-- 从ChEMBL 35 提取特定靶标的活性数据
-- 为了演示目的，限制为100 行。
-- 避免numeric ~ 运算符错误。
SELECT
    act.molregno,
    act.standard_type,
    act.standard_relation,
    act.standard_value,
    act.standard_units,
    cmpd.smiles
FROM activities act
JOIN assays ass ON act.assay_id = ass.assay_id
JOIN target_dictionary td ON ass.tid = td.tid
JOIN molecule_dictionary md ON act.molregno = md.molregno
JOIN compound_structures cmpd ON md.molregno = cmpd.molregno
WHERE td.chembl_id = 'CHEMBL206'  -- 替换成你的目标靶标的ChEMBL ID
  AND act.standard_type = 'IC50' --筛选IC50 值
  AND act.standard_value IS NOT NULL -- 排除空值
  AND act.standard_value != '' -- 排除空字符串
```

```
  AND act.standard_value::TEXT ~ '^[0-9\.]+$' -- 确保数值仅包含数字
LIMIT 100;
```

Important: Replace `'CHEMBL206'` with the actual ChEMBL ID of the target you're interested in. You can search for target IDs on the ChEMBL website.

How to run:

1. **pgAdmin:** Open pgAdmin, connect to your `chembl_35` database.
2. **New Query:** Open a new query window.
3. **Paste SQL:** Paste the SQL code above into the query window.
4. **Execute:** Execute the query.
5. **Save as CSV:** Right-click on the result grid and select "Copy with Headers". Then, paste the copied data into a text editor and save it as a `.csv` file (e.g., `Topic_CheMBL_35_76_data.csv`) in the `data/` directory. Alternatively, use the "Export..." option to directly save the query results as a CSV file.

2. Python Code (English & Chinese)

This Python code (intended for a Jupyter Notebook) reads the CSV file, preprocesses the data, calculates some basic RDKit descriptors, and demonstrates a simple model.

```
# English
import pandas as pd
import os
from rdkit import Chem
from rdkit.Chem import Descriptors
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np

# Define the base path for the project (adjust as needed)
base_path = '.'  # Assuming the notebook is in the project root

# Construct the path to the CSV file
csv_file_path = os.path.join(base_path, 'data', 'Topic_CheMBL_35_76_data.csv')

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

# Data cleaning and preparation
df = df.dropna(subset=['smiles', 'standard_value'])
df = df[df['standard_value'] != '']
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce') #Convert
to number
df = df.dropna(subset=['standard_value']) #Drop NA after converting to number
# RDKit Mol object creation
df['mol'] = df['smiles'].apply(lambda x: Chem.MolFromSmiles(x))
df = df.dropna(subset=['mol'])  # Remove rows where Mol object creation failed

# Feature calculation (example: Molecular Weight)
df['mol_weight'] = df['mol'].apply(lambda x: Descriptors.MolWt(x))

# Prepare data for modeling
```

```
X = df[['mol_weight']] #Features
y = df['standard_value'] #Target

# 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)

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

# Prediction
y_pred = model.predict(X_test)

# Evaluation
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}")



# Chinese
import pandas as pd
import os
from rdkit import Chem
from rdkit.Chem import Descriptors
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np

# 定义项目的基础路径（根据需要进行调整）
base_path = '.'  # 假设notebook 在项目根目录中

# 构建CSV 文件的路径
csv_file_path = os.path.join(base_path, 'data', 'Topic_CheMBL_35_76_data.csv')

# 加载数据
try:
    df = pd.read_csv(csv_file_path)
except FileNotFoundError:
    print(f"错误：在{csv_file_path} 未找到文件。请确保已运行SQL 查询并保存数据。")
    exit()

# 数据清理和准备
df = df.dropna(subset=['smiles', 'standard_value'])
df = df[df['standard_value'] != '']
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce') #Convert
to number
df = df.dropna(subset=['standard_value']) #Drop NA after converting to number
# 创建RDKit Mol 对象
df['mol'] = df['smiles'].apply(lambda x: Chem.MolFromSmiles(x))
df = df.dropna(subset=['mol'])  # 删除Mol 对象创建失败的行

# 特征计算（示例：分子量）
df['mol_weight'] = df['mol'].apply(lambda x: Descriptors.MolWt(x))

# 准备用于建模的数据
```

```
X = df[['mol_weight']] # 特征
y = df['standard_value'] # 目标变量

# 将数据分成训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 模型训练
model = LinearRegression()
model.fit(X_train, y_train)

# 预测
y_pred = model.predict(X_test)

# 评估
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
print(f"均方误差 (MSE): {mse}")
print(f"均方根误差 (RMSE): {rmse}")
```

How to run:

1. **Create Notebook:** Create a new Jupyter Notebook named `Topic_CheMBL_35_76_2_data_preprocessing.ipynb` (or a suitable name following your naming convention) in the `notebooks/` directory.
2. **Paste Code:** Paste the Python code above into the notebook.
3. **Run Cells:** Execute the code cells one by one.
4. **Check Output:** Examine the output to ensure the code is running correctly.

Important Notes:

- **Install Libraries:** Make sure you have the necessary libraries installed: `pandas`, `rdkit`, `scikit-learn`, `numpy`. You can install them using `pip install pandas rdkit scikit-learn numpy`. If you encounter issues installing RDKit via pip, follow the instructions on the RDKit website for your specific operating system.
- **File Paths:** Double-check the `base_path` and `csv_file_path` to ensure they correctly point to your data file.
- **Error Handling:** The code includes basic error handling (e.g., checking for file existence). Expand this to handle other potential errors, such as invalid SMILES strings.
- **Data Exploration:** Before modeling, thoroughly explore your data. Look at distributions of `standard_value`, relationships between features, and potential outliers.
- **Feature Engineering:** The example code only uses molecular weight as a feature. Experiment with other RDKit descriptors and features relevant to your target and the biological activity.
- **Model Selection:** Linear regression is a very basic model. Explore more advanced machine learning models suitable for your data (e.g., Random Forest, Support Vector Machines, neural networks).

5 Examples

Here are five examples of things you could do *after* running the above code, building on the foundation. Each example is a small extension, illustrating a different aspect of the analysis.

1. **Calculate More Descriptors:** Add more molecular descriptors using RDKit (e.g., LogP, TPSA, number of rings). Modify the `feature calculation` and `prepare data for modeling` parts of the python code above to include the extra descriptors. This allows you to build a more complex model.

```
from rdkit.Chem import Crippen, Lipinski, rdMolDescriptors

df['logp'] = df['mol'].apply(lambda x: Crippen.MolLogP(x))
df['tpsa'] = df['mol'].apply(lambda x: rdMolDescriptors.CalcTPSA(x))
df['num_rings'] = df['mol'].apply(lambda x: Lipinski.RingCount(x))

X = df[['mol_weight', 'logp', 'tpsa', 'num_rings']] #More features
```

2. **Visualize Data:** Create a scatter plot of `mol_weight` vs. `standard_value` to visually inspect the relationship. This can help you assess the suitability of linear regression and identify potential outliers.

```
import matplotlib.pyplot as plt

plt.scatter(df['mol_weight'], df['standard_value'])
plt.xlabel('Molecular Weight')
plt.ylabel('Standard Value (IC50)')
plt.title('Molecular Weight vs. IC50')
plt.show()
```

3. **Log Transform Activity:** Apply a log transformation to the `standard_value` if it's highly skewed. This can improve the performance of some models.

```
df['log_standard_value'] = np.log10(df['standard_value'])
y = df['log_standard_value']  # Target variable with log transformed data
```

4. **Use a Different Model:** Train a Random Forest Regressor instead of Linear Regression. This often provides better performance, especially with non-linear relationships.

```
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)
rmse = np.sqrt(mse)
print(f"Random Forest Mean Squared Error: {mse}")
print(f"Random Forest Root Mean Squared Error: {rmse}")
```

5. **Cross-Validation:** Use k-fold cross-validation to get a more robust estimate of the model's performance.

```
from sklearn.model_selection import cross_val_score

model = LinearRegression()  # or RandomForestRegressor
scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_error')  #
5-fold cross-validation
rmse_scores = np.sqrt(-scores)  # Convert negative MSE to RMSE
print("Cross-validation RMSE scores:", rmse_scores)
print("Mean cross-validation RMSE:", rmse_scores.mean())
```

Remember to adjust the code and analysis based on your specific research goals and the characteristics of your data. Good luck!