PharmApp Suite

🧠 AI for Drug Discovery and Development 🍼

🧩 Topic: Mining antimicrobial targets in ChEMBL

Okay, I understand. You're working with ChEMBL 35 data, using RDKit, and following an AI/MLOps template structure for your project. You have a PostgreSQL database on a remote server, and your code will run in Jupyter Notebooks. You've encountered specific errors and want to filter the data for efficiency. Let's break down the analysis and provide the code, along with some examples.

1. Analysis of the Analysis Model & Errors

The core of your task is to extract and analyze data related to drug activity from the ChEMBL database. This typically involves:

- **Data Extraction:** Querying the ChEMBL database (using SQL) to retrieve relevant information about compounds and their bioactivity data. This involves joining various tables like `activities`, `assays`, `compound_structures`, etc.
- **Data Preprocessing:** Cleaning and transforming the extracted data for machine learning or data analysis. This includes:
    - Filtering for specific activity types (e.g., IC50, Ki).
    - Converting activity values to a consistent unit (e.g., pIC50).
    - Handling missing values.
    - Converting SMILES strings to RDKit Mol objects.
- **Feature Generation:** Calculating molecular descriptors using RDKit. These descriptors represent the chemical properties of the compounds and are used as features for machine learning models.
- **Model Building (if applicable):** Training a machine learning model to predict activity based on the molecular descriptors. Common models include regression (e.g., Linear Regression, Random Forest) for predicting continuous activity values or classification (e.g., Logistic Regression, Support Vector Machines) for predicting active/inactive status.
- **Analysis and Interpretation:** Evaluating the performance of the model (if applicable) and interpreting the results to gain insights into structure-activity relationships (SAR).

Error Analysis:

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

  This error means that the PostgreSQL database does not recognize the ~ operator (regex match) for a numeric column (`act.standard_value`). PostgreSQL requires you to cast the numeric column to text before performing a regular expression match.

- **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. The `squared=False` parameter was introduced in a later version to return the Root Mean Squared Error (RMSE) instead of the Mean Squared Error (MSE). You have two options:

  1. **Upgrade Scikit-learn:** This is the preferred solution. `pip install -U scikit-learn`
  2. **Calculate RMSE manually:** If upgrading is not feasible, calculate the square root of the MSE. `rmse = np.sqrt(mean_squared_error(y_true, y_pred))`

2. SQL and Python Code (English)

Here's the SQL and Python code, adhering to your file structure and incorporating the error fixes.

SQL (Save as `../data/chembl_35_query.sql`)

```sql
-- chembl_35_query.sql
SELECT
    cs.canonical_smiles,
    act.standard_value,
    act.standard_units,
    act.activity_comment,
    md.chembl_id AS molecule_chembl_id,
    td.chembl_id AS target_chembl_id,
    td.target_type
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 cs ON md.molregno = cs.molregno
WHERE
    td.target_type = 'SINGLE PROTEIN'
    AND act.standard_type = 'IC50'
    AND act.standard_units = 'nM'
    AND act.standard_value IS NOT NULL
    AND act.standard_value::TEXT ~ '^[0-9\.]+$'  -- Corrected regex: casting to TEXT
    AND act.standard_value <= 10000   -- Filter for reasonable IC50 values
LIMIT 100;

-- You would then use pgAdmin to run this SQL script and save the result as a CSV
file, e.g., chembl_35_data.csv in the ../data/ directory
```

Python (Jupyter Notebook - `notebook/Topic_CheMBL_35_37_1_data_extraction.ipynb`)

```python
# notebook/Topic_CheMBL_35_37_1_data_extraction.ipynb
import os
import pandas as pd
import numpy as np
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, r2_score
import matplotlib.pyplot as plt

# Define base path
base_path = ".."  # Assuming notebook is in the 'notebook' directory

# Data loading and preprocessing
data_path = os.path.join(base_path, "data", "chembl_35_data.csv")
df = pd.read_csv(data_path)

# Handle missing values (drop rows with missing values)
df = df.dropna()

# Convert standard_value to numeric and handle errors
```

```
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')
df = df.dropna(subset=['standard_value'])

# Convert IC50 to pIC50
df['pIC50'] = -np.log10(df['standard_value'] / 1e9)  # Convert nM to M, then -log10

# RDKit Feature Generation
def generate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None
    descriptors = {}
    for name, func in Descriptors.descList:
        try:
            descriptors[name] = func(mol)
        except:
            descriptors[name] = np.nan  # Handle errors gracefully
    return pd.Series(descriptors)

# Apply descriptor generation, handling potential errors in SMILES
df_descriptors = df['canonical_smiles'].apply(generate_descriptors)

# Concatenate descriptors to original DataFrame
df = pd.concat([df, df_descriptors], axis=1)

# Drop rows with any NaN values resulting from descriptor calculation
df = df.dropna()

# Data Splitting (Example - adjust as needed)
X = df.drop(['canonical_smiles', 'standard_value', 'standard_units',
'activity_comment', 'molecule_chembl_id', 'target_chembl_id', 'target_type', 'pIC50'],
axis=1)
y = df['pIC50']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Model Training (Linear Regression Example)
model = LinearRegression()
model.fit(X_train, y_train)

# Prediction
y_pred = model.predict(X_test)

# Evaluation
mse = mean_squared_error(y_test, y_pred) # Calculate MSE
rmse = np.sqrt(mse) # Calculate RMSE

r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}") # Print RMSE
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 of the Python code:

1. **Imports:** Imports necessary libraries (os, pandas, RDKit, scikit-learn, matplotlib).
2. **Path Definition:** Defines the base path and constructs the full path to the data file using `os.path.join`.
3. **Data Loading:** Reads the CSV file into a Pandas DataFrame.
4. **Data Cleaning:** Handles missing values by dropping rows using `df.dropna()`. Converts `standard_value` to numeric and removes any resulting NaN values.
5. **pIC50 Conversion:** Converts IC50 values (in nM) to pIC50 values.
6. **Feature Generation (RDKit):**
    - Defines a function `generate_descriptors` that takes a SMILES string as input and calculates a set of molecular descriptors using RDKit.
    - Applies this function to the 'canonical_smiles' column of the DataFrame. Error handling is included within the function to deal with potentially invalid SMILES strings. The error handling in the descriptor generation is extremely important. If a single SMILES string fails, the entire process can halt.
    - The generated descriptors are added as new columns to the DataFrame.
    - Rows with NaN descriptors are dropped to ensure data integrity.
7. **Data Splitting:** Splits the data into training and testing sets.
8. **Model Training:** Trains a Linear Regression model on the training data.
9. **Prediction:** Uses the trained model to predict pIC50 values for the test data.
10. **Evaluation:** Calculates the Mean Squared Error (MSE), Root Mean Squared Error (RMSE) and R-squared.
11. **Visualization:** Creates a scatter plot of predicted vs. actual pIC50 values.

3. Examples (within a new notebook: `notebook/Topic_CheMBL_35_37_2_analysis_examples.ipynb`)

Here are five examples of how you might use this data:

```
# notebook/Topic_CheMBL_35_37_2_analysis_examples.ipynb

import os
import pandas as pd
import numpy as np
from rdkit import Chem
from rdkit.Chem import Descriptors
import matplotlib.pyplot as plt
import seaborn as sns

# Define base path
base_path = ".."  # Assuming notebook is in the 'notebook' directory

# Data loading
data_path = os.path.join(base_path, "data", "chembl_35_data.csv")
df = pd.read_csv(data_path)

# Preprocessing (same as before, important to have consistent data)
df = df.dropna()
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')
df = df.dropna(subset=['standard_value'])
df['pIC50'] = -np.log10(df['standard_value'] / 1e9)

def generate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None
```

```
    descriptors = {}
    for name, func in Descriptors.descList:
        try:
            descriptors[name] = func(mol)
        except:
            descriptors[name] = np.nan  # Handle errors gracefully
    return pd.Series(descriptors)

# Apply descriptor generation, handling potential errors in SMILES
df_descriptors = df['canonical_smiles'].apply(generate_descriptors)

# Concatenate descriptors to original DataFrame
df = pd.concat([df, df_descriptors], axis=1)

# Drop rows with any NaN values resulting from descriptor calculation
df = df.dropna()


# Example 1: Distribution of pIC50 values
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()


# Example 2: Correlation between Molecular Weight and pIC50
plt.figure(figsize=(8, 6))
plt.scatter(df['MolWt'], df['pIC50'])
plt.xlabel('Molecular Weight (MolWt)')
plt.ylabel('pIC50')
plt.title('Correlation between Molecular Weight and pIC50')
plt.show()

# Example 3: Top 10 most frequent targets
target_counts = df['target_chembl_id'].value_counts().head(10)
plt.figure(figsize=(10, 6))
target_counts.plot(kind='bar')
plt.title('Top 10 Most Frequent Targets')
plt.xlabel('Target ChEMBL ID')
plt.ylabel('Frequency')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

# Example 4: Distribution of a specific descriptor (e.g., LogP)
plt.figure(figsize=(8, 6))
sns.histplot(df['MolLogP'], kde=True)
plt.title('Distribution of LogP Values')
plt.xlabel('LogP')
plt.ylabel('Frequency')
plt.show()

# Example 5: Scatter plot of two descriptors
plt.figure(figsize=(8, 6))
plt.scatter(df['MolWt'], df['MolLogP'])
plt.xlabel('Molecular Weight (MolWt)')
plt.ylabel('LogP')
```

```
plt.title('Scatter Plot of Molecular Weight vs. LogP')
plt.show()
```

Explanation of the Examples:

- **Example 1: Distribution of pIC50 values:** Visualizes the distribution of the target variable (pIC50) to understand its range and central tendency.
- **Example 2: Correlation between Molecular Weight and pIC50:** Explores the relationship between a simple molecular descriptor (Molecular Weight) and activity.
- **Example 3: Top 10 most frequent targets:** Identifies the most frequently studied targets in the dataset.
- **Example 4: Distribution of a specific descriptor (e.g., LogP):** Visualizes the distribution of a specific molecular property.
- **Example 5: Scatter plot of two descriptors:** Explores the relationship between two molecular properties.

Key Considerations and Best Practices:

- **Error Handling:** The code includes error handling during SMILES parsing and descriptor calculation. This is *crucial* for real-world datasets, as not all SMILES strings will be valid.
- **Data Filtering:** The SQL query filters for specific activity types and units. Adjust these filters to match your specific research question. It also limits the initial query to 100 rows.
- **Virtual Environment:** Always use a virtual environment for your Python projects to manage dependencies and avoid conflicts.
- **Data Exploration:** Spend time exploring your data before building models. Understand the distributions of your variables, identify outliers, and look for potential biases.
- **Feature Selection:** Consider using feature selection techniques to reduce the dimensionality of your data and improve model performance.
- **Model Validation:** Use appropriate validation techniques (e.g., cross-validation) to evaluate the performance of your models.
- **Documentation:** Document your code thoroughly. Explain the purpose of each step and the rationale behind your choices.
- **Reproducibility:** Make your work reproducible by using version control (e.g., Git) and by clearly specifying the versions of all software packages used.

Two Languages (English and Chinese):

Here's a summary in both English and Chinese:

English Summary:

You are analyzing ChEMBL 35 data using RDKit, following an AI/MLOps template. The process involves extracting data from a PostgreSQL database, cleaning and preprocessing it, generating molecular descriptors with RDKit, and potentially building machine learning models to predict activity. Key challenges include handling errors in SMILES parsing, managing database connections, and using appropriate evaluation metrics. The provided code includes an SQL query to extract data, a Python script for data preprocessing and feature generation, and example analyses. Remember to use virtual environments, document your code, and handle potential errors gracefully.

Chinese Summary:

您正在使用 RDKit 分析 ChEMBL 35 数据，并遵循 AI/MLOps 模板。这个过程包括从 PostgreSQL 数据库中提取数据，进行数据清洗和预处理，使用 RDKit 生成分子描述符，并可能构建机器学习模型来预测活性。主要的挑战包括处理 SMILES 解析中的错误，管理数据库连接，以及使用适当的评估指标。提供的代码包括用于提取数据的 SQL 查询，用于数据预处理和特征生成的 Python 脚本，以及一些分析示例。请记住使用虚拟环境，记录您的代码，并妥善处理潜在的错误。

Let me know if you have more questions or need further assistance!