PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: Radar chart of drug-likeness scores

Okay, I understand. You're working with ChEMBL 35 data, using RDKit, and following an AIMLops-like directory structure. You need guidance on running your SQL and Python code, handling errors, and creating examples. Let's break this down.

Analysis of the Analysis Model (Assuming a Regression Task)

Based on the context of ChEMBL data (bioactivity data), the `standard_value` field, and common use cases, I'm going to assume you're building a *regression model* to predict bioactivity. The specific bioactivity being predicted (e.g., IC50, Ki) would depend on the filtering in your SQL query.

The general workflow will be:

1. **Data Extraction (SQL):** Retrieve relevant data from the ChEMBL database, focusing on specific targets or assays. The SQL query will filter for data where the `standard_value` is a numerical value and limit the results to 100 rows to avoid overwhelming your system. The query will save the output to a CSV file.
2. **Data Preprocessing (Python/RDKit):** Load the data from the CSV file. Use RDKit to generate molecular descriptors (e.g., fingerprints, physicochemical properties) from the SMILES strings in the ChEMBL database.
3. **Model Building and Evaluation (Python/Scikit-learn):** Split the data into training and testing sets. Train a regression model (e.g., Random Forest, linear regression) using the molecular descriptors as features and the `standard_value` as the target variable. Evaluate the model's performance using metrics like Mean Squared Error (MSE) or R-squared.

Addressing Errors

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

  This error occurs because you are trying to use a regular expression operator `~` on a numeric column (`act.standard_value`). PostgreSQL doesn't allow this directly. You need to cast the numeric column to a text column before applying the regular expression.

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

  This means you're using an older version of scikit-learn (likely pre 0.20). There are two solutions:

  - **Upgrade scikit-learn:** This is the preferred solution: `pip install -U scikit-learn`
  - **Remove `squared=False`:** If upgrading isn't feasible, calculate the Root Mean Squared Error (RMSE) manually by taking the square root of the MSE: `rmse = np.sqrt(mean_squared_error(y_test, y_pred))`

Directory Structure (Following AIMLops)

Let's assume the following base path:

```
base_path = "/path/to/your/project"  # Replace with your actual path
```

Then, we have:

- `data/`: Contains the extracted CSV file.
- `notebooks/`: Contains the Jupyter Notebooks.
- `models/`: (Will contain saved models later).
- `src/`: (Optional; could contain reusable Python modules).
- `config/`: (Optional; could contain configuration files).

1. SQL Code (to extract data)

```sql
-- File: Topic_CheMBL_35_79_extract.sql
-- Saved in: ../data/Topic_CheMBL_35_79_data.csv

COPY (
    SELECT
        act.molregno,
        act.standard_value,
        md.chembl_id,
        cs.canonical_smiles
    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 for IC50 values
        AND act.standard_units = 'nM' -- Ensure units are consistent
        AND act.standard_value IS NOT NULL
        AND act.standard_value::text ~ '^[0-9\.]+$'  -- Corrected: Cast to text for
regex
    LIMIT 100  -- Limit to 100 rows
) TO '/path/to/your/project/data/Topic_CheMBL_35_79_data.csv' WITH CSV HEADER;

-- Change '/path/to/your/project/data/Topic_CheMBL_35_79_data.csv' to actual folder
```

Important Considerations for SQL:

- **Adjust the path in the COPY command:** Make sure the path `/path/to/your/project/data/Topic_CheMBL_35_79_data.csv` is accessible to the PostgreSQL server (it needs write permissions). If you have issues, consider using `\COPY` from `psql` instead, which writes to the client's filesystem.
- **`standard_type`:** Critically, choose the correct `standard_type` (e.g., 'IC50', 'Ki', 'EC50'). This determines what kind of bioactivity you are predicting.
- **Units:** Standardize the units (`standard_units`). 'nM' is common, but verify in your data.
- **Bioactivity Range:** Consider adding filtering for a reasonable range of `standard_value`. Extremely high or low values can be outliers. For example: `AND act.standard_value BETWEEN 0 AND 100000` (adjust range as needed).
- **Target Specificity:** Ideally, filter for a specific target (e.g., a specific protein). This makes the model more relevant. You'll need to join with the `target_dictionary` table and use `target_components` and `component_sequences` tables to find your target. This is a more advanced query.

Running the SQL (in pgAdmin):

1. Open pgAdmin and connect to your `chembl_35` database (ip: 192.168.206.136, user: rd, pass: rd).
2. Open a query window.
3. Paste the SQL code into the query window.

4. **Important:** Edit the `/path/to/your/project/data/Topic_CheMBL_35_79_data.csv` part to reflect the *actual* path on the *server* where PostgreSQL is running.
5. Execute the query.

2. Python Code (Jupyter Notebook)

```
# File: notebooks/Topic_CheMBL_35_79_1_data_prep.ipynb

import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import AllChem
from rdkit.Chem import Descriptors
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import joblib # For saving the model

# Define the base path
base_path = "/path/to/your/project"  # Replace with your actual path
data_path = os.path.join(base_path, "data", "Topic_CheMBL_35_79_data.csv")
model_path = os.path.join(base_path, "models", "Topic_CheMBL_35_79_model.joblib")


# 1. Load the data
try:
    df = pd.read_csv(data_path)
    print("Data loaded successfully.")
except FileNotFoundError:
    print(f"Error: File not found at {data_path}.  Make sure you ran the SQL query and
that the path is correct.")
    exit()

# 2. Data Cleaning and Preparation
df = df.dropna(subset=['canonical_smiles', 'standard_value'])  # Drop rows with
missing SMILES or values
df = df[df['standard_value'] > 0]  # Remove non-positive standard values

# 3. RDKit Feature Generation (Molecular Descriptors)
def generate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None  # Handle invalid SMILES

    # Calculate Descriptors (Example: Molecular Weight, LogP)
    mw = Descriptors.MolWt(mol)
    logp = Descriptors.MolLogP(mol)

    # Calculate Morgan Fingerprint (ECFP4)
    fingerprint = AllChem.GetMorganFingerprintAsBitVect(mol, radius=2, nBits=2048)
    fingerprint_array = np.array(list(fingerprint.ToBitString()), dtype=int)

    return pd.Series([mw, logp, *fingerprint_array]) # Combine into a single series

# Apply descriptor generation to each SMILES string
descriptors = df['canonical_smiles'].apply(generate_descriptors)

# Handle cases where descriptor generation failed (invalid SMILES)
```

```
df = df.join(descriptors).dropna()

# Separate features (X) and target (y)
X = df.iloc[:, 5:] # Select columns from position 5 to the end because first 5 columns
were about ID, SMILE, etc.
y = df['standard_value']

# 4. Data Splitting
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 5. Model Training
model = RandomForestRegressor(n_estimators=100, random_state=42)  # Example: Random
Forest
model.fit(X_train, y_train)

# 6. Model Evaluation
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mse)

print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")
print(f"Root Mean Squared Error: {rmse}")

# 7. Save the Model
joblib.dump(model, model_path)
print(f"Model saved to {model_path}")
```

File: notebooks/Topic_CheMBL_35_79_2_model_inference.ipynb

```
# File: notebooks/Topic_CheMBL_35_79_2_model_inference.ipynb

import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import AllChem
from rdkit.Chem import Descriptors
import numpy as np
import joblib # For loading the model

# Define the base path
base_path = "/path/to/your/project"  # Replace with your actual path
data_path = os.path.join(base_path, "data", "Topic_CheMBL_35_79_data.csv")
model_path = os.path.join(base_path, "models", "Topic_CheMBL_35_79_model.joblib")

# 1. Load the Model
try:
    model = joblib.load(model_path)
    print("Model loaded successfully.")
except FileNotFoundError:
    print(f"Error: Model file not found at {model_path}.  Make sure you ran the
training notebook first.")
    exit()

# 2. Load New Data (or a subset of the original data)
try:
    df = pd.read_csv(data_path)
    print("Data loaded successfully.")
```

```
except FileNotFoundError:
    print(f"Error: File not found at {data_path}.  Make sure you ran the SQL query and
that the path is correct.")
    exit()

# 3. Data Cleaning and Preparation
df = df.dropna(subset=['canonical_smiles', 'standard_value'])  # Drop rows with
missing SMILES or values
df = df[df['standard_value'] > 0]  # Remove non-positive standard values

# 4. RDKit Feature Generation (Molecular Descriptors)
def generate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None  # Handle invalid SMILES

    # Calculate Descriptors (Example: Molecular Weight, LogP)
    mw = Descriptors.MolWt(mol)
    logp = Descriptors.MolLogP(mol)

    # Calculate Morgan Fingerprint (ECFP4)
    fingerprint = AllChem.GetMorganFingerprintAsBitVect(mol, radius=2, nBits=2048)
    fingerprint_array = np.array(list(fingerprint.ToBitString()), dtype=int)

    return pd.Series([mw, logp, *fingerprint_array]) # Combine into a single series

# Apply descriptor generation to each SMILES string
descriptors = df['canonical_smiles'].apply(generate_descriptors)

# Handle cases where descriptor generation failed (invalid SMILES)
df = df.join(descriptors).dropna()

# Separate features (X) and target (y)
X = df.iloc[:, 5:] # Select columns from position 5 to the end because first 5 columns
were about ID, SMILE, etc.

# 5. Make Predictions
predictions = model.predict(X)

# 6. Add Predictions to the DataFrame
df['predicted_value'] = predictions

# 7. Display Results (First 10 rows)
print(df[['chembl_id', 'standard_value', 'predicted_value']].head(10))
```

Key Points for Python Code:

- **Error Handling:** The `try...except` block handles the `FileNotFoundError` when loading the CSV. Add more error handling, especially around the RDKit descriptor generation (invalid SMILES strings are common).
- **RDKit Descriptor Generation:** This is a *critical* step. The code provides a basic example (molecular weight and LogP), and Morgan fingerprints. Experiment with other descriptors. Consider using the `rdkit.Chem.Descriptors` module for more physicochemical properties. Be consistent with number of fingerprints with the number of columns you select.
- **Model Choice:** Random Forest is a good starting point. Experiment with other regression models like linear regression, Support Vector Regression (SVR), or Gradient Boosting.

- **Hyperparameter Tuning:** Optimize the hyperparameters of your chosen model (e.g., `n_estimators`, `max_depth` for Random Forest). Use techniques like cross-validation and grid search.
- **Feature Selection:** Not all descriptors will be equally important. Use feature selection techniques to reduce the dimensionality of the data and improve model performance. The model's `feature_importances_` attribute can help with this.
- **Logging:** Use the `logging` module for more robust logging than just `print` statements.

3. Examples

Here are 5 examples of what you can do with this setup:

1. **Predict IC50 for a specific target:** Modify the SQL query to filter for a specific target (e.g., a specific protein or enzyme) using `target_dictionary` and related tables. Train a model to predict IC50 values for inhibitors of that target.
2. **Compare different descriptor sets:** Experiment with different sets of RDKit descriptors (e.g., physicochemical properties, topological indices, fingerprints). Train separate models with each descriptor set and compare their performance.
3. **Evaluate different machine learning models:** Compare the performance of different regression models (e.g., Random Forest, linear regression, SVR) on the same dataset and feature set.
4. **Build a QSAR model for a series of compounds:** If you have a set of related compounds (e.g., analogs) with known activities, build a QSAR (Quantitative Structure-Activity Relationship) model to predict the activity of new compounds in the series.
5. **Implement cross-validation:** Use cross-validation to get a more robust estimate of your model's performance. This will help you avoid overfitting to the training data.

English Explanation

This setup allows you to extract bioactivity data from the ChEMBL database, generate molecular descriptors using RDKit, and build machine learning models to predict bioactivity. By modifying the SQL query and experimenting with different descriptors and models, you can explore the relationships between chemical structure and biological activity, which is essential for drug discovery. The AIMLops-style directory structure helps you organize your code and data. Remember to handle errors carefully and use best practices for model building and evaluation.

Explanation in Indonesian

Pengaturan ini memungkinkan Anda untuk mengekstrak data bioaktivitas dari database ChEMBL, menghasilkan deskriptor molekul menggunakan RDKit, dan membangun model machine learning untuk memprediksi bioaktivitas. Dengan memodifikasi query SQL dan bereksperimen dengan deskriptor dan model yang berbeda, Anda dapat menjelajahi hubungan antara struktur kimia dan aktivitas biologis, yang penting untuk penemuan obat. Struktur direktori ala AIMLops membantu Anda mengatur kode dan data Anda. Ingatlah untuk menangani kesalahan dengan hati-hati dan menggunakan praktik terbaik untuk membangun dan mengevaluasi model.