PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: Tanimoto similarity search using ChEMBL + RDKit

Okay, I understand. As a Pharmaceutical Research and Development Specialist, I'll help you structure your Chembl 35 data analysis project using RDKit, addressing your specific requirements and the errors you've encountered. I'll provide SQL and Python code snippets, aiming for a streamlined and reproducible workflow within your AIMLops template structure.

1. Analysis Model

The core of this project involves extracting relevant chemical and biological data from the ChEMBL database, cleaning it, transforming it, and then applying machine learning techniques to identify potential relationships between molecular properties and biological activity. The goal is to extract information that can be used for drug design.

Here's a breakdown:

- **Data Extraction (SQL):** We'll use SQL to query the ChEMBL database to extract compound information (e.g., SMILES strings) and bioactivity data (e.g., IC50 values) for a specific target or a set of targets. The SQL code will also handle filtering to limit the number of rows returned (as you requested, ~100 rows) and address the numeric comparison error you reported.
- **Data Preparation (Python):**
    - **RDKit for Molecular Representation:** RDKit will be used to generate molecular fingerprints (e.g., Morgan fingerprints, also known as ECFP) from the SMILES strings. These fingerprints are numerical representations of molecular structure, suitable for machine learning algorithms.
    - **Bioactivity Data Transformation:** Bioactivity data (like IC50) often needs to be transformed (e.g., converted to pIC50) to provide a more suitable scale for modeling. We will perform this transformation.
    - **Data Cleaning:** Handling missing values and outliers is crucial for model performance.
- **Model Building (Python):**
    - **Feature Selection:** We'll explore different feature selection methods to identify the most relevant molecular properties for predicting activity.
    - **Model Selection:** We'll use a supervised learning technique. This could be a simple regression model (e.g., Linear Regression) or a more complex model (e.g., Random Forest, Support Vector Machine). Given the small dataset size, Linear Regression or a simpler tree-based model might be a good starting point.
    - **Model Evaluation:** We'll split the data into training and test sets and use appropriate metrics (e.g., Mean Squared Error, R-squared) to evaluate the model's performance.

2. Code (SQL & Python)

SQL Code (saved as `../data/chembl_35_9.sql`):

```sql
-- Select 100 rows for demonstration purposes
SELECT
    cmpd.chembl_id,
    cmpd.pref_name,
    act.standard_value,
    act.standard_units,
    act.standard_type,
    act.activity_comment,
```

```
    act.relation,
    md.molecule_structures AS smiles
FROM
    compound_structures md
JOIN
    molecule_dictionary cmpd ON md.molregno = cmpd.molregno
JOIN
    activities act ON cmpd.molregno = act.molregno
JOIN
    target_dictionary tgt ON act.tid = tgt.tid
WHERE
    tgt.target_type = 'SINGLE PROTEIN'
    AND tgt.pref_name = 'Dopamine D2 receptor'
    AND act.standard_type = 'IC50'
    AND act.standard_units = 'nM'
    AND act.standard_value IS NOT NULL  -- Ensure standard_value is not null
    AND act.standard_value::TEXT ~ '^[0-9\.]+$'  -- Check if standard_value is numeric
ORDER BY random()
LIMIT 100;

-- Copy the result to a CSV file (run this in psql after connecting to the database)
\copy (SELECT
    cmpd.chembl_id,
    cmpd.pref_name,
    act.standard_value,
    act.standard_units,
    act.standard_type,
    act.activity_comment,
    act.relation,
    md.molecule_structures AS smiles
FROM
    compound_structures md
JOIN
    molecule_dictionary cmpd ON md.molregno = cmpd.molregno
JOIN
    activities act ON cmpd.molregno = act.molregno
JOIN
    target_dictionary tgt ON act.tid = tgt.tid
WHERE
    tgt.target_type = 'SINGLE PROTEIN'
    AND tgt.pref_name = 'Dopamine D2 receptor'
    AND act.standard_type = 'IC50'
    AND act.standard_units = 'nM'
    AND act.standard_value IS NOT NULL  -- Ensure standard_value is not null
    AND act.standard_value::TEXT ~ '^[0-9\.]+$'  -- Check if standard_value is numeric
ORDER BY random()
LIMIT 100) TO '../data/chembl_35_9.csv' DELIMITER ',' CSV HEADER;
```

Explanation of SQL Code:

- **`SELECT ... FROM ... JOIN ... WHERE ...`**: This is a standard SQL query to select data from multiple tables. We are joining `compound_structures`, `molecule_dictionary`, `activities`, and `target_dictionary` tables to get compound information, bioactivity data, and target information.
- **`tgt.target_type = 'SINGLE PROTEIN'`**: This filters for single protein targets.
- **`tgt.pref_name = 'Dopamine D2 receptor'`**: This filters specifically for the Dopamine D2 receptor. You can change this to any target you're interested in.
- **`act.standard_type = 'IC50'`**: This filters for IC50 values.
- **`act.standard_units = 'nM'`**: This filters for IC50 values in nanomolar units.

- **`act.standard_value IS NOT NULL`**: Crucially, this avoids missing values, which cause errors in later steps.
- **`AND act.standard_value::TEXT ~ '^[0-9\.]+$'`**: This part addresses the error you encountered. It converts the `standard_value` column to text and uses a regular expression (`^[0-9\.]+$`) to check if the value contains only numbers and decimal points. This ensures that you're only working with numeric values. The type conversion to `TEXT` and the use of `~` operator performs the intended filtering.
- **`ORDER BY random()`**: This randomizes the results.
- **`LIMIT 100`**: This limits the result set to 100 rows.
- **`\copy ... TO ...`**: This is a psql command to copy the results of the query directly to a CSV file. Make sure you run this command *within* the psql terminal after connecting to your database. This avoids having to export the data manually from pgAdmin.

Python Code (`notebook/Topic_CheMBL_35_9_1.ipynb`):

```
import os
import pandas as pd
import numpy as np
from rdkit import Chem
from rdkit.Chem import AllChem
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 math  # Import the math module

# Define the base path
base_path = ".." # Assuming notebook is in the 'notebook' directory
data_path = os.path.join(base_path, "data")
csv_file = os.path.join(data_path, "chembl_35_9.csv")

# Load the data
try:
    df = pd.read_csv(csv_file)
    print(f"Data loaded successfully from {csv_file}")
except FileNotFoundError:
    print(f"Error: File not found at {csv_file}.  Make sure you've run the SQL and
copied the data.")
    exit()

# Data Cleaning and Transformation
df = df.dropna(subset=['standard_value', 'smiles']) # Drop rows with missing values in
these columns

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

# RDKit Molecular Fingerprint Generation
def generate_fingerprint(smiles):
    try:
        mol = Chem.MolFromSmiles(smiles)
        if mol is not None:
            fp = AllChem.GetMorganFingerprintAsBitVect(mol, 2, nBits=2048)  # ECFP4
fingerprint
            return np.array(fp)
        else:
            return None
```

```
    except:
        return None

df['fingerprint'] = df['smiles'].apply(generate_fingerprint)
df = df.dropna(subset=['fingerprint']) # Drop rows where fingerprint generation failed

# Prepare data for machine learning
X = np.vstack(df['fingerprint'].values)
y = df['pIC50'].values

# 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)

# Train a Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

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


# Example Predictions
print("\nExample Predictions:")
for i in range(5): # Show predictions for the first 5 compounds in the test set
    print(f"Actual pIC50: {y_test[i]:.2f}, Predicted pIC50: {y_pred[i]:.2f}")
```

Explanation of Python Code:

- **Import Libraries:** Imports necessary libraries, including pandas for data manipulation, RDKit for cheminformatics, and scikit-learn for machine learning.
- **File Path Handling:** Uses `os.path.join` to construct file paths, making the code more robust and portable.
- **Data Loading:** Loads the CSV file into a pandas DataFrame. Includes error handling for the case where the file is not found.
- **Data Cleaning:** Drops rows with missing SMILES strings or standard values.
- **pIC50 Conversion:** Calculates pIC50 values from IC50 values. Note the conversion from nM to M before taking the negative logarithm.
- **Fingerprint Generation:** Defines a function `generate_fingerprint` to generate ECFP4 fingerprints using RDKit. It handles potential errors in SMILES parsing.
- **Data Preparation:** Converts the fingerprint column to a NumPy array suitable for scikit-learn.
- **Train/Test Split:** Splits the data into training and test sets.
- **Model Training:** Creates a Linear Regression model and trains it on the training data.
- **Model Evaluation:** Calculates the Mean Squared Error and R-squared on the test data.
- **Example Predictions:** Prints the actual and predicted pIC50 values for the first 5 compounds in the test set.

Addressing the Error in Mean Squared Error (older scikit-learn versions):

The `squared=False` parameter in `mean_squared_error` was introduced in a later version of scikit-learn. If you are using an older version, you can calculate the Root Mean Squared Error (RMSE) manually:

```
mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)  # Use the math module to calculate the square root
print(f"Root Mean Squared Error: {rmse}")
```

3. Examples (based on the code above):

Here are 5 examples of things you can do with this code, reflecting common tasks in drug discovery:

1. **Change the Target:** Modify the SQL query to target a different protein. For example, change `tgt.pref_name = 'Dopamine D2 receptor'` to `tgt.pref_name = 'EGFR'`. You'll need to find the correct target name in the ChEMBL database.
2. **Change the Activity Type:** Instead of IC50, you might be interested in Ki values. Change `act.standard_type = 'IC50'` to `act.standard_type = 'Ki'`. You'll also need to adjust the pIC50 calculation accordingly if the units are different.
3. **Change the Fingerprint Type:** Experiment with different RDKit fingerprints. For example, try using `Chem.RDKFingerprint(mol)` instead of `AllChem.GetMorganFingerprintAsBitVect(mol, 2, nBits=2048)`. You'll need to research the different fingerprint types and their parameters.
4. **Try a Different Model:** Replace the `LinearRegression()` model with a different model, such as `RandomForestRegressor()` or `SVR()`. You'll need to import the corresponding class from scikit-learn and potentially tune the model's hyperparameters.
5. **Feature Selection:** Before training the model, try using feature selection techniques to reduce the dimensionality of the fingerprint data. Scikit-learn provides various feature selection methods, such as `SelectKBest` or `RFE` (Recursive Feature Elimination).

Important Considerations:

- **Database Connection:** Make sure you have a working connection to your ChEMBL database and that the user `rd` has the necessary permissions to query the data.
- **RDKit Installation:** Ensure that RDKit is installed correctly in your Jupyter Notebook environment.
- **Dependencies:** Install all the necessary Python libraries (pandas, numpy, scikit-learn, rdkit). You can use `pip install pandas numpy scikit-learn rdkit`.
- **Error Handling:** The code includes basic error handling, but you may need to add more robust error handling for production use.
- **Documentation:** Document your code thoroughly, explaining the purpose of each step and the rationale behind your choices. This will make it easier to maintain and extend the code in the future.

In Summary (en français):

Ce projet vous permet d'extraire des données de la base de données ChEMBL, de préparer ces données avec RDKit (pour la représentation moléculaire), et d'appliquer des techniques d'apprentissage automatique pour explorer les relations entre les propriétés moléculaires et l'activité biologique. Le code SQL vous aide à récupérer les données pertinentes, et le code Python utilise RDKit pour générer des empreintes digitales moléculaires et scikit-learn pour construire et évaluer des modèles prédictifs. Les exemples fournis vous montrent comment modifier le code pour cibler différentes protéines, types d'activité, et algorithmes d'apprentissage automatique. N'oubliez pas de vérifier vos connexions de base de données, d'installer toutes les dépendances nécessaires et de documenter soigneusement votre code.

Let me know if you have any further questions. Good luck with your project!