PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: Khai thác mối quan hệ cấu trúc – hoạt tính (SAR)

Tuyệt vời! Tôi rất vui được hỗ trợ bạn trong dự án phân tích dữ liệu Chembl 35. Dựa trên thông tin bạn cung cấp, chúng ta sẽ tập trung vào chủ đề **Topic_CheMBL_35_8**.

1. Phân tích mô hình phân tích (Analysis of Analytical Model):

Chủ đề này có thể liên quan đến việc khám phá mối quan hệ giữa cấu trúc hóa học của các hợp chất (từ Chembl 35) và hoạt tính sinh học của chúng. Chúng ta có thể sử dụng RDKit để tính toán các descriptor phân tử (ví dụ: trọng lượng phân tử, LogP, số lượng liên kết quay được) từ cấu trúc SMILES của các hợp chất. Sau đó, chúng ta có thể sử dụng các mô hình học máy (ví dụ: hồi quy tuyến tính, Random Forest, SVM) để dự đoán hoạt tính sinh học dựa trên các descriptor này.

Here's a breakdown of the analytical model:

- **Data Source:** Chembl 35 database (accessed via PostgreSQL).
- **Data Extraction:** SQL queries to retrieve compound structures (SMILES) and bioactivity data (e.g., IC50 values).
- **Feature Engineering:** RDKit to calculate molecular descriptors from SMILES strings.
- **Model Building:** Scikit-learn (or other ML libraries) to train predictive models.
- **Model Evaluation:** Metrics like R-squared, RMSE, etc., to assess model performance.

2. Hướng dẫn song ngữ (Bilingual Guidance):

SQL (English):

We will use SQL to extract relevant data from the Chembl 35 database. This includes compound IDs, SMILES strings, and bioactivity measurements.

SQL (Tiếng Việt):

Chúng ta sẽ sử dụng SQL để trích xuất dữ liệu liên quan từ cơ sở dữ liệu Chembl 35. Dữ liệu này bao gồm ID hợp chất, chuỗi SMILES và các phép đo hoạt tính sinh học.

Python (English):

Python will be used for data preprocessing, feature engineering (using RDKit), model building, and evaluation. Libraries like pandas, RDKit, scikit-learn, and matplotlib will be essential.

Python (Tiếng Việt):

Python sẽ được sử dụng để tiền xử lý dữ liệu, tính toán đặc trưng (sử dụng RDKit), xây dựng mô hình và đánh giá. Các thư viện như pandas, RDKit, scikit-learn và matplotlib sẽ rất quan trọng.

3. Code SQL, Python tiếng Anh (SQL, Python Code in English):

Here are examples of SQL and Python code snippets to get you started.

SQL (Example 1: Extracting Data):

```sql
-- Lấy 100 dòng dữ liệu hoạt tính sinh học với IC50 từ Chembl
-- Get 100 rows of bioactivity data with IC50 from Chembl
SELECT
    cmp.chembl_id,
    cmp.canonical_smiles,
```

```
    act.standard_value,
    act.standard_units
FROM
    compound_structures cmp
JOIN
    activities act ON cmp.molregno = act.molregno
JOIN
    assay_components ac ON act.assay_id = ac.assay_id
JOIN
    target_dictionary td ON ac.tid = td.tid
WHERE
    act.standard_type = 'IC50'
    AND act.standard_units = 'nM'
    AND act.standard_value IS NOT NULL
    AND act.standard_value > 0 -- Loại bỏ giá trị âm hoặc bằng 0
    AND cmp.canonical_smiles IS NOT NULL
    AND act.standard_value::text ~ '^[0-9\.]+$' -- Lọc các giá trị không phải số
LIMIT 100;
```

Python (Example 1: Calculate Molecular Descriptors):

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

base_path = "." # Thay đổi nếu cần thiết

# Đọc dữ liệu từ file CSV đã lưu
# Read data from the saved CSV file
csv_file_path = os.path.join(base_path, "data", "chembl_ic50_100.csv")  # Thay đổi tên
file nếu cần
df = pd.read_csv(csv_file_path)

def calculate_descriptors(smiles):
    """Calculates molecular descriptors using RDKit."""
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None  # Handle invalid SMILES strings
    descriptors = {
        "mol_weight": Descriptors.MolWt(mol),
        "logp": Descriptors.MolLogP(mol),
        "num_h_donors": Descriptors.NumHDonors(mol),
        "num_h_acceptors": Descriptors.NumHAcceptors(mol),
    }
    return descriptors

# Áp dụng hàm tính toán descriptor cho cột SMILES
# Apply the descriptor calculation function to the SMILES column
df['descriptors'] = df['canonical_smiles'].apply(calculate_descriptors)

# Tách các descriptors vào các cột riêng biệt
# Separate the descriptors into separate columns
df = pd.concat([df, df['descriptors'].apply(pd.Series)], axis=1)

# Loại bỏ các hàng có giá trị descriptor bị thiếu (do SMILES không hợp lệ)
# Remove rows with missing descriptor values (due to invalid SMILES)
df = df.dropna(subset=['mol_weight', 'logp', 'num_h_donors', 'num_h_acceptors'])
```

```
print(df.head())
```

4. Sửa lỗi (Error Correction):

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

  Lỗi này xảy ra vì bạn đang cố gắng sử dụng toán tử ~ (regular expression match) trên một cột kiểu `numeric`. Để khắc phục, hãy cast cột `standard_value` sang kiểu `text` trước khi so sánh:

  SQL (Corrected):

  ```
  AND act.standard_value::text ~ '^[0-9\.]+$'
  ```

- **Lỗi Python (Python Error):** `squared=False`

  Nếu bạn gặp lỗi này, có nghĩa là phiên bản scikit-learn bạn đang sử dụng quá cũ. Hãy nâng cấp scikit-learn lên phiên bản mới hơn (>= 0.20) bằng lệnh:

  ```
  pip install --upgrade scikit-learn
  ```

  Hoặc, nếu không muốn nâng cấp, hãy bỏ tham số `squared=False` và lấy căn bậc hai của kết quả `mean_squared_error` để tính RMSE:

  ```
  from sklearn.metrics import mean_squared_error
  import numpy as np

  mse = mean_squared_error(y_true, y_pred)
  rmse = np.sqrt(mse) # Calculate RMSE
  ```

5. Ví dụ code .sql và python mẫu (Sample SQL and Python Code):

Here are 5 examples to get you started:

SQL Examples:

1. **Extract Compound IDs and SMILES:**

   ```
   SELECT chembl_id, canonical_smiles FROM compound_structures LIMIT 100;
   ```

2. **Extract IC50 values for a specific target (example: target ID 'CHEMBL205'):**

   ```
   SELECT act.standard_value
   FROM activities act
   JOIN assay_components ac ON act.assay_id = ac.assay_id
   WHERE ac.tid = (SELECT tid FROM target_dictionary WHERE chembl_id = 'CHEMBL205')
   AND act.standard_type = 'IC50'
   AND act.standard_units = 'nM'
   LIMIT 100;
   ```

3. **Find compounds with molecular weight less than 500:** `sql SELECT cmp.chembl_id, cmp.canonical_smiles FROM compound_structures cmp WHERE cmp.molweight < 500 LIMIT 100;`

4. **Join compound structures and activities tables based on molregno:**

   ```
   SELECT cmp.chembl_id, cmp.canonical_smiles, act.standard_value
   FROM compound_structures cmp
   JOIN activities act ON cmp.molregno = act.molregno
   LIMIT 100;
   ```

5. **Extract data for a specific assay (example: assay ID 666715):**
```sql
SELECT act.molregno, act.standard_value
FROM activities act
WHERE act.assay_id = 666715
AND act.standard_type = 'IC50'
AND act.standard_units = 'nM'
AND act.standard_value IS NOT NULL
LIMIT 100;
```

Python Examples:

1. **Calculate a single descriptor (Molecular Weight):**

```python
from rdkit import Chem
from rdkit.Chem import Descriptors

smiles = 'CC(=O)Oc1ccccc1C(=O)O'
mol = Chem.MolFromSmiles(smiles)
mol_weight = Descriptors.MolWt(mol)
print(f"Molecular Weight: {mol_weight}")
```

2. **Load SMILES from a DataFrame and calculate LogP:**

```python
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors

# Assuming you have a DataFrame 'df' with a column 'canonical_smiles'
def calculate_logp(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol:
        return Descriptors.MolLogP(mol)
    else:
        return None

df['logp'] = df['canonical_smiles'].apply(calculate_logp)
print(df[['canonical_smiles', 'logp']].head())
```

3. **Train a Linear Regression Model:**

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

# Assuming you have a DataFrame 'df' with 'standard_value' (IC50) and molecular descriptors
# Convert IC50 to pIC50
df = df[df['standard_value'].notna()]
df['pIC50'] = -np.log10(df['standard_value']*1e-9) # Convert nM to Molar

X = df[['mol_weight', 'logp', 'num_h_donors', 'num_h_acceptors']]
y = df['pIC50']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()
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"RMSE: {rmse}")
```

4. **Handle Missing SMILES:**

```
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors

def calculate_descriptors_safe(smiles):
    try:
        mol = Chem.MolFromSmiles(smiles)
        if mol is None:
            return None
        descriptors = {
            "mol_weight": Descriptors.MolWt(mol),
            "logp": Descriptors.MolLogP(mol),
        }
        return descriptors
    except:
        return None

df['descriptors'] = df['canonical_smiles'].apply(calculate_descriptors_safe)
df = df.dropna(subset=['descriptors'])  # Remove rows with invalid SMILES
```

5. **Visualize the results**

```
import matplotlib.pyplot as plt
# Assuming you have y_test and y_pred from the model
plt.scatter(y_test, y_pred)
plt.xlabel("Actual pIC50")
plt.ylabel("Predicted pIC50")
plt.title("Actual vs Predicted pIC50 Values")
plt.show()
```

Important Considerations:

- **Data Cleaning:** Chembl data can be noisy. Always clean and validate your data before modeling. This includes handling missing values, outliers, and incorrect SMILES strings.
- **Feature Selection:** Experiment with different molecular descriptors. Feature selection techniques can help improve model performance.
- **Model Selection:** Try different machine learning models (e.g., Random Forest, SVM, Gradient Boosting) to see which performs best.
- **Cross-Validation:** Use cross-validation to get a more robust estimate of model performance.
- **pIC50:** Convert IC50 values to pIC50 (pIC50 = -log10(IC50 in M)) for better distribution and modeling. Remember to convert your IC50 values to Molar before taking the logarithm.

Tôi sẵn sàng giúp bạn giải quyết các vấn đề cụ thể hơn khi bạn tiến triển trong dự án. Chúc bạn thành công!