Untitled

March 28, 2025

[9]:
```python
import numpy as np
import pandas as pd
from rdkit import Chem
from rdkit.Chem import AllChem
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
```

[2]:
```python
ds= pd.read_csv("KIBA.csv")
ds =ds.rename(columns={"Ki , Kd and IC50  (KIBA Score)":"binding_affinity"})
ds= ds.sample(n=20000, random_state=42)
print(ds.head())
```

```
            CHEMBLID ProteinID  \
58637  CHEMBL1981744    P49760
7623    CHEMBL185238    Q15759
28019  CHEMBL1997597    O14757
72021  CHEMBL1682546    P49760
85578  CHEMBL2005528    P52333

                                        compound_iso_smiles  \
58637  C1=CC(=NC(=C1)NC(=O)NC2=C3C=CC(=CC3=NC=C2)Cl)C…
7623      CC1=CC=CC(=N1)C2=C(C=NN2)C3=NC4=C(C=C3)N=CC=C4
28019      CCS(=O)(=O)NC1=CC=C(C=C1)C2=CC3=C(C=C2)NN=C3N
72021      C1=CC=C(C=C1)CNC2=NC(=CS2)C3=CC4=C(C=C3)NN=C4
85578                CCOC(=O)C1=CC2=C(C=C1)NC3=C2CCNC3=O

                                          target_sequence  binding_affinity
58637  MPHPRRYHSSERGSRGSYREHYRSRKHKRRRSRSWSSSSDRTRRRR…         11.500000
7623   MSGPRAGFYRQELNKTVWEVPQRLQGLRPVGSGAYGSVCSAYDARL…         10.895880
28019  MAVPFVEDWDLVQTLGEGAYGEVQLAVNRVTEEAVAVKIVDMKRAV…         11.200000
72021  MPHPRRYHSSERGSRGSYREHYRSRKHKRRRSRSWSSSSDRTRRRR…         11.800000
85578  MAPPSEETPLIPQRSCSLLSTEAGALHVLLPARGPGPPQRLSFSFG…         11.900001
```

[3]:
```python
ds= ds.dropna()
print(ds.isna().sum())  # This should print 0 for all columns
```

```
CHEMBLID               0
```

```
ProteinID             0
compound_iso_smiles   0
target_sequence       0
binding_affinity      0
dtype: int64
```

0.1 Preprocessing the Dataset

[4]:
```
#1) for kiba score
threshold=np.mean(ds['binding_affinity'])
ds['target'] = (ds['binding_affinity'] > threshold).astype(int)
print(ds.head())
```

```
            CHEMBLID ProteinID  \
58637  CHEMBL1981744    P49760
7623    CHEMBL185238    Q15759
28019  CHEMBL1997597    O14757
72021  CHEMBL1682546    P49760
85578  CHEMBL2005528    P52333

                                   compound_iso_smiles  \
58637  C1=CC(=NC(=C1)NC(=O)NC2=C3C=CC(=CC3=NC=C2)Cl)C…
7623      CC1=CC=CC(=N1)C2=C(C=NN2)C3=NC4=C(C=C3)N=CC=C4
28019      CCS(=O)(=O)NC1=CC=C(C=C1)C2=CC3=C(C=C2)NN=C3N
72021      C1=CC=C(C=C1)CNC2=NC(=CS2)C3=CC4=C(C=C3)NN=C4
85578                CCOC(=O)C1=CC2=C(C=C1)NC3=C2CCNC3=O

                                       target_sequence  binding_affinity  \
58637  MPHPRRYHSSERGSRGSYREHYRSRKHKRRRSRSWSSSSDRTRRRR…         11.500000
7623   MSGPRAGFYRQELNKTVWEVPQRLQGLRPVGSGAYGSVCSAYDARL…         10.895880
28019  MAVPFVEDWDLVQTLGEGAYGEVQLAVNRVTEEAVAVKIVDMKRAV…         11.200000
72021  MPHPRRYHSSERGSRGSYREHYRSRKHKRRRSRSWSSSSDRTRRRR…         11.800000
85578  MAPPSEETPLIPQRSCSLLSTEAGALHVLLPARGPGPPQRLSFSFG…         11.900001

       target
58637       0
7623        0
28019       0
72021       1
85578       1
```

[5]:
```
def smiles_to_fingerprint(smiles, radius=2, n_bits=1024):
    molecule = Chem.MolFromSmiles(smiles)
    if molecule is not None:
        # Use AllChem.GetMorganFingerprintAsBitVect to generate a fingerprint
        fingerprint = AllChem.GetMorganFingerprintAsBitVect(molecule,␣
↪radius=radius, nBits=n_bits)
        return list(fingerprint)
    else:
        return [0] * n_bits  # Return a zero vector if the SMILES is invalid

# Apply the function to generate the fingerprints for each compound in the␣
↪dataset
smiles_fingerprints = ds['compound_iso_smiles'].apply(smiles_to_fingerprint)

# Instead of creating individual columns for each fingerprint bit, create a␣
↪single vector column
ds['compound_iso_smiles'] = smiles_fingerprints

# Now the 'fingerprints' column contains the fingerprint vectors as a single␣
↪list in each row
print(ds.head())
```

```
            CHEMBLID ProteinID  \
58637  CHEMBL1981744    P49760
7623    CHEMBL185238    Q15759
28019  CHEMBL1997597    O14757
72021  CHEMBL1682546    P49760
85578  CHEMBL2005528    P52333

                                     compound_iso_smiles  \
58637  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
7623   [0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
28019  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
72021  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
85578  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …

                                         target_sequence  binding_affinity  \
58637  MPHPRRYHSSERGSRGSYREHYRSRKHKRRRSRSWSSSSDRTRRRR…         11.500000
7623   MSGPRAGFYRQELNKTVWEVPQRLQGLRPVGSGAYGSVCSAYDARL…         10.895880
28019  MAVPFVEDWDLVQTLGEGAYGEVQLAVNRVTEEAVAVKIVDMKRAV…         11.200000
72021  MPHPRRYHSSERGSRGSYREHYRSRKHKRRRSRSWSSSSDRTRRRR…         11.800000
85578  MAPPSEETPLIPQRSCSLLSTEAGALHVLLPARGPGPPQRLSFSFG…         11.900001

       target
58637       0
7623        0
28019       0
72021       1
85578       1
```

[6]:
```
def strip_sequence(seq, length=50):
    # Truncate to the first 'length' characters
    return seq[:length].ljust(length, 'X')  # padding with 'X' if the sequence␣
↪is shorter than 50
```

```
# Apply the function to the protein sequence column
ds['target_sequence'] = ds['target_sequence'].apply(strip_sequence)

# Print the updated DataFrame
print(ds.head())
```

```
              CHEMBLID ProteinID  \
58637   CHEMBL1981744    P49760
7623     CHEMBL185238    Q15759
28019   CHEMBL1997597    O14757
72021   CHEMBL1682546    P49760
85578   CHEMBL2005528    P52333

                                     compound_iso_smiles  \
58637   [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
7623    [0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
28019   [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
72021   [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
85578   [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …

                                         target_sequence  binding_affinity  \
58637   MPHPRRYHSSERGSRGSYREHYRSRKHKRRRSRSWSSSSDRTRRRR…         11.500000
7623    MSGPRAGFYRQELNKTVWEVPQRLQGLRPVGSGAYGSVCSAYDARL…         10.895880
28019   MAVPFVEDWDLVQTLGEGAYGEVQLAVNRVTEEAVAVKIVDMKRAV…         11.200000
72021   MPHPRRYHSSERGSRGSYREHYRSRKHKRRRSRSWSSSSDRTRRRR…         11.800000
85578   MAPPSEETPLIPQRSCSLLSTEAGALHVLLPARGPGPPQRLSFSFG…         11.900001

       target
58637       0
7623        0
28019       0
72021       1
85578       1
```

[7]:
```
# Define the standard 20 amino acids
amino_acids = ['A', 'R', 'N', 'D', 'C', 'Q', 'E', 'G', 'H', 'I', 'L', 'K', 'M',␣
 ↪'F', 'P', 'S', 'T', 'W', 'Y', 'V']

# Function to perform one-hot encoding for a single protein sequence
def one_hot_encode(sequence, amino_acids):
    # Initialize a list to hold the one-hot encoded vectors
    one_hot_encoded = []

    # Iterate over each character in the sequence
    for aa in sequence:
        # Create a binary vector of length 20, where 1 represents the presence␣
 ↪of that amino acid
        encoding = [1 if aa == amino_acid else 0 for amino_acid in amino_acids]
        one_hot_encoded.append(encoding)

    # Return the list of one-hot encoded vectors as a numpy array
    return np.array(one_hot_encoded)

# Sample KIBA dataset (Replace this with your actual `ds` DataFrame)
# For demonstration purposes, I'll create a small subset.
# Replace the following with your actual KIBA dataset (20,000 rows with␣
 ↪'target_sequence' column)

# Apply one-hot encoding to the 'target_sequence' column
ds['target_sequence'] = ds['target_sequence'].apply(lambda seq:␣
 ↪one_hot_encode(seq, amino_acids))

# Check the result
print(ds['target_sequence'][0].shape)  # Output the shape of the first one-hot␣
 ↪encoded sequence

# Display the DataFrame with one-hot encoded sequences (first few rows)
print(ds.head())
# Save the DataFrame to a CSV file
ds.to_csv('output_dataset.csv', index=False)  # 'index=False' prevents pandas␣
 ↪from writing row numbers (index) to the file
```

```
(50, 20)
            CHEMBLID ProteinID  \
58637  CHEMBL1981744    P49760
7623    CHEMBL185238    Q15759
28019  CHEMBL1997597    O14757
72021  CHEMBL1682546    P49760
85578  CHEMBL2005528    P52333

                                        compound_iso_smiles  \
58637  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
7623   [0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
28019  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
72021  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
85578  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …

                                            target_sequence  binding_affinity  \
58637  [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,…         11.500000
7623   [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,…         10.895880
28019  [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,…         11.200000
72021  [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,…         11.800000
```

```
85578  [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,…         11.900001

       target
58637       0
7623        0
28019       0
72021       1
85578       1
```

[8]:
```
from sklearn.decomposition import PCA

X_smiles = np.array(ds['compound_iso_smiles'].tolist())  # Assumes␣
↪'compound_iso_smiles' is a list of bits (0s, 1s)
X_sequence = np.array(ds['target_sequence'].tolist())  # Assumes␣
↪'target_sequence' is a list of 50x20 matrices

# Concatenate the features: flatten target_sequence (50, 20) into a 1D vector␣
↪and concatenate with X_smiles
X = np.hstack((X_smiles, X_sequence.reshape(X_sequence.shape[0], -1)))  #␣
↪Flatten and concatenate

# Extract target labels (binary classification: 0 or 1)
y = ds['target'].values  # Binary target (0 or 1)

# Step 6: Train-test split (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Step 7: Feature scaling (important for SVM)
scaler = StandardScaler()

# Scale the data
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 8: Train the SVM model
svm_model = SVC(kernel='linear', random_state=42)  # You can use other kernels␣
↪like 'rbf' or 'poly' too

# Train the model on the reduced data
svm_model.fit(X_train_scaled, y_train)

# Step 9: Evaluate the model
y_pred = svm_model.predict(X_test_scaled)

# Print evaluation metrics
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.81775
Classification Report:
               precision    recall  f1-score   support

           0       0.83      0.88      0.86      2468
           1       0.79      0.72      0.75      1532

    accuracy                           0.82      4000
   macro avg       0.81      0.80      0.80      4000
weighted avg       0.82      0.82      0.82      4000
```

```
[ ]:
```

```
[ ]:
```