Project: Drug-Target Binding Affinity Estimation using Cross-Validation Techniques

Dear Data Scientist,

I have a long-term research project regarding a specific set of proteins. I am attempting to discover small organic compounds that can bind strongly to these proteins and thus act as drugs. I have already made laboratory experiments to measure the affinities between some proteins and drug molecules.

My colleague is working on another set of proteins, and the objectives of his project are similar to mine. He has recently discovered thousands of new potential drug molecules. He asked me if I could find the pairs that have the strongest affinities among his proteins and drug molecules. Obviously I do not have the resources to measure all the possible pairs in my laboratory, so I need to prioritise. I decided to do this with the help of machine learning, but I have encountered a problem.

Here is what I have done so far: First I trained a K-nearest neighbours regressor with the parameter value K=10 using all the 400 measurements I had already made in the laboratory with my proteins and drug molecules. They comprise of 77 target proteins and 59 drug molecules. Then I performed a leave-one-out cross-validation with this same data to estimate the generalisation performance of the model. I used C-index and got a stellar score above 90%. Finally I used the model to predict the affinities of my colleague's proteins and drug molecules. The problem is: when I selected the highest predicted affinities and tried to verify them in the lab, I found that many of them are much lower in reality. My model clearly does not work despite the high cross-validation score.

Please explain why my estimation failed and how leave-one-out cross-validation should be performed to get a reliable estimate. Also, implement the correct leave-one-out cross-validation and report its results. I need to know whether it would be a waste of my resources if I were to use my model any further.

The data I used to create my model is available in the files `input.data`, `output.data` and `pairs.data` for you to use. The first file contains the features of the pairs, whereas the second contains their affinities. The third file contains the identifiers of the drug and target molecules of which the pairs are composed. The files are paired, i.e. the i^{th} row in each file is about the same pair.

Looking forward to hearing from you soon.

Yours sincerely,
Bio Scientist

Try to answer the questions about cross-validation on pair-input data

In [7]:
```
# Why did the estimation described in the letter fail?
# How should leave-one-out cross-validation be performed in the given scenario
# Remember to provide comprehensive and precise arguments.
```

Why did the estimation described in the letter fail?

Answer: The leave-one-out cross-validation (LOOCV) used by the researcher failed because it did not account for the structure of the data, which lead to data leakage. Since the following dataset consists of protein-drug pairs, removing only one pair at a time means that the same proteins and drugs still appear in both the training and validation sets. This will allow the K-Nearest Neighbors (KNN) model to rely on previously seen proteins or drugs data samples for making predictions, which is creating an illusion of high accuracy. The model does not truly generalize but instead memorizes affinity values for specific proteins and drugs, which explains the exceptionally high cross-validation score (C-index > 90%). However, when the model was applied to an entirely new dataset with unseen proteins and drugs, its predictions turned out to be unreliable. This failure shows that the model was overfitting to the known proteins and drugs in the dataset rather than learning a general pattern for predicting affinities.

How should leave-one-out cross-validation be performed in the given scenario and why?

Answer To obtain a better and reliable estimate of generalization performance, I think cross-validation should be designed in such a way that ensures no proteins or drugs in the validation set appear in the training set. Instead of standard LOOCV, we could use Leave-Protein-Out (LPO) cross-validation or Leave-Pair-Out (LPO) cross-validation. In Leave-Protein-Out CV, each fold removes all pairs associated with a specific protein (or drug) so that the model cannot rely on prior knowledge of that protein when making predictions. Similarly, Leave-Pair-Out CV removes specific protein-drug interactions across different folds to avoid any overlap. These approaches prevent data leakage and ensure that the model is tested only on truly unseen data. An alternative method could be Grouped K-Fold Cross-Validation, where proteins or drugs are grouped together and assigned to distinct folds, ensuring no mixing between training and validation sets. These techniques provide a more realistic estimate of how well the model would perform in a real-world scenario where it must predict affinities for entirely new proteins and drugs.

Why will the correct cross-validation method give a reliable estimate of generalization performance?

Answer: I think the correct cross-validation method will provide a more accurate estimate of generalization performance because it eliminates possibility of data leakage and prevents overfitting to specific proteins or drugs. When proteins and drugs are removed entirely from the training set before being tested, the given model is forced to make predictions based on general patterns rather than memorized information gained during training. This will simulate the real-world challenge of predicting affinities for unseen protein-drug pairs, which is the main goal of the research. I think by ensuring that validation data is truly independent from training data, the revised approach will yield performance metrics that reflect how well the model will work on new experimental data.

Import libraries

In [24]:
```
# Import the libraries you need.
import numpy as np
import pandas as pd
from sklearn.model_selection import GroupKFold, LeaveOneGroupOut
from sklearn.neighbors import KNeighborsRegressor
from scipy.stats import pearsonr
from itertools import combinations
import random
```

Write utility functions

In [9]:
```
# Write the utility functions you need in your analysis.
```

In [10]:
```
def calculate_c_index(y_true, y_pred):
    """Compute the Concordance Index (C-Index) to evaluate model performance."
    n = 0 # Total number of comparable pairs
    h_sum = 0.0 # Count of correctly ranked pairs
    for i in range(len(y_true)):
        for j in range(i + 1, len(y_true)):
            if y_true[i] != y_true[j]: # Consider only pairs with different tr
                n += 1
                h_sum += int((y_pred[i] > y_pred[j]) == (y_true[i] > y_true[j]
    return h_sum / n if n > 0 else 0.5 # Return C-Index value, defaulting to 0

def evaluate_model(y_true, y_pred):
    """Evaluate the model using Pearson correlation and C-Index."""
    pearson_corr, _ = pearsonr(y_true, y_pred)
    c_index = calculate_c_index(y_true, y_pred)
    return {
        "Pearson Correlation": pearson_corr,
        "C-Index": c_index
    }
```

Load datasets

In [11]:
```
# Read the data files (input.data, output.data, pairs.data).
```

In [12]:
```
input_data = pd.read_csv("input.data", delim_whitespace=True, header=None)
output_data = pd.read_csv("output.data", delim_whitespace=True, header=None)
pairs_data = pd.read_csv("pairs.data", delim_whitespace=True, header=None)
```

In [13]:
```
# Display first few rows to confirm successful loading
input_data.head()
```

Out[13]:

	0	1	2	3	4	5	6	7	8	
0	0.759222	0.709585	0.253151	0.421082	0.727780	0.404487	0.709027	0.242963	0.407292	0.3
1	0.034584	0.304720	0.688257	0.296396	0.151878	0.830755	0.270656	0.705392	0.186120	0.0
2	0.737867	0.236079	0.905987	0.163612	0.801455	0.789823	0.393999	0.522067	0.411352	0.7
3	0.406913	0.607740	0.235365	0.888679	0.150347	0.598991	0.130108	0.465818	0.799953	0.9
4	0.697707	0.432565	0.650329	0.886065	0.328660	0.576926	0.523100	0.080463	0.131349	0.9

5 rows × 67 columns

In [14]:
```
# Display first few rows to confirm successful loading

output_data.head()
```

Out[14]:

	0
0	0.733933
1	0.569419
2	0.832588
3	0.389664
4	0.725953

In [15]:
```
# Display first few rows to confirm successful loading
pairs_data.head()
```

Out[15]:

	0	1
0	D40	T2
1	D31	T64
2	D6	T58
3	D56	T49
4	D20	T28

Implement and run cross-validation

In [16]:
```
# Implement and run the requested cross-validation. Report and interpret its r
```

In [25]:
```
# Wanted to see and implementing Leave-Pair-Out Cross-Validation
def perform_optimized_leave_pair_out(X, y, pairs, sample_fraction=0.05):
    """Perform Leave-Pair-Out cross-validation with randomized subsampling for
    Since LPO is computationally expensive, we use subsampling to reduce the n
    Instead of evaluating all possible pairs, we randomly sample a fraction of
    This maintains representative evaluation while making the process more pra
    """
    predictions = np.zeros(len(y))

    pair_indices = list(combinations(range(len(y)), 2))
    sampled_pairs = random.sample(pair_indices, int(len(pair_indices) * sample

    for idx1, idx2 in sampled_pairs:
        train_idx = [i for i in range(len(y)) if i not in (idx1, idx2)]
        test_idx = [idx1, idx2]

        model = KNeighborsRegressor(n_neighbors=10)
        model.fit(X.iloc[train_idx], y.iloc[train_idx])
        predictions[test_idx] = model.predict(X.iloc[test_idx])

    return evaluate_model(y, predictions)
```

In [26]:
```
cv_lpo_results = perform_optimized_leave_pair_out(X, y, pairs_data, sample_fra
print("Optimized Leave-Pair-Out Cross-Validation Results:", cv_lpo_results)
```

```
Optimized Leave-Pair-Out Cross-Validation Results: {'Pearson Correlation':
0.8687611484332883, 'C-Index': 0.8302255639097744}
```

In [29]:
```
#Implementing Grouped Validation
def perform_grouped_cross_validation(X, y, groups, n_splits=5):
    """Perform grouped k-fold cross-validation ensuring no data leakage.
    This method ensures that all data points related to a specific group (e.g.
    are kept together in either the training or test set, avoiding information
    gkf = GroupKFold(n_splits=n_splits)
    predictions = np.zeros(len(y))

    for train_idx, test_idx in gkf.split(X, y, groups):
        model = KNeighborsRegressor(n_neighbors=10)
        model.fit(X.iloc[train_idx], y.iloc[train_idx])
        predictions[test_idx] = model.predict(X.iloc[test_idx])

    return evaluate_model(y, predictions)
```

In [20]:
```
# Extract features, target, and groups (proteins as groups)
X = input_data
y = output_data[0]
groups = pairs_data[1]  # Assuming proteins are in column index 1
```

In [28]:
```
# Run Grouped K-Fold cross-validation and report results
cv_grouped_results = perform_grouped_cross_validation(X, y, groups)
print("Grouped K-Fold Cross-Validation Results:", cv_grouped_results)
```

```
Grouped K-Fold Cross-Validation Results: {'Pearson Correlation': 0.810478353
2220297, 'C-Index': 0.7942606516290727}
```

Implications and Interpretations of Results

The cross-validation results indicate that the model has a Pearson correlation of 0.8105, showing a strong positive relationship between predicted and actual affinity values, and a Concordance Index (C-Index) of 0.7943, which suggests that the model is reasonably good at ranking protein-drug pairs based on binding strength. Compared to the previous overoptimistic estimation (~90% C-Index) caused by data leakage in previous case, the new results provide a more realistic assessment of the model's generalization ability. While the model can be useful for prioritizing strong affinity pairs, it should not be blindly trusted for definitive predictions without further improvements.