Université de Genève

–

Sciences Informatiques

13X011 Data Mining
Molecular Property Prediction

June 2024

Contents

Molecular Property Prediction

1 Introduction

Aim

In this project, you are going to work with molecular data and your goal is to predict the "pIC50" - a measurement used in pharmacology and drug discovery to assess the potency of a compound in inhibiting a specific biological target or enzyme - as well as the "logP" - a measure of how a molecule interacts with different solvents. It indicates the molecule's preference for nonpolar (oily) or polar (water-like) environments.

Dataset

The given dataset (data.csv) consists of a set of formatted values for 16,087 molecules/SMILES and their associated pI50 and logP values.
You are also given the number of atoms in each molecule, but this number can be easily recovered from the SMILES representation.

In this project, we use Random Forest to predict two molecular properties: pIC50 and logP. To evaluate the performance of our Random Forest model, we compare it against a baseline algorithm. This comparison helps us understand the added value of using a more complex model like Random Forest.

1.1 Data Structure

Shape of the data:

This is what our data looks like

SMILES	pIC50	num_atoms	logP
O=S(=O)(Nc1cccc(-c2cnc3ccccc3n2)c1)cccs1	4.26	25	4.1591
O=c1cc(-c2nc(-c3ccc(-c4cn(CCP(=O)(O)O)nn4)cc3)...	4.34	36	3.6743
NC(=O)c1ccc2c(c1)nc(C1CCC(O)CC1)n2CCCO	4.53	23	1.5361
NCCCn1c(C2CCNCC2)nc2cc((N)=O)ccc21	4.56	22	0.9510
CNC(=S)Nc1cccc(-c2cnc3ccccc3n2)c1	4.59	21	3.2130

And from this we are going to extract several other features.
Namely:

- Number of C
- Number of O
- Number of N
- Number of atoms
- Number of bonds
- Number of rotatable bonds: The number of bonds which allow free rotation around themselves
- Number of rings: The number of groups of atoms and bonds where each atom and bond is part of a loop or ring structure.
- Molecular weight: The weight of a molecule based on the atomic masses of all atoms in the molecule
- Number of H Donors
- Number of H Acceptors

These are the features we're going to use to predict the pIC50 and logP values.

This is its behavior according to `pandas.describe()`:

	pIC50	num_atoms	logP
count	15037.000000	16087.000000	16087.000000
mean	0.998739	18.749984	2.256600
std	2.479588	8.428888	1.609861
min	0.000000	3.000000	-5.395600
25%	0.000000	13.000000	1.214900
50%	0.010000	17.000000	2.163200
75%	0.130000	23.000000	3.232440
max	10.970000	85.000000	15.879200

1.2 Data Cleaning

To extract the aforementionned features, we will use the RDKit library and implement the parsing *SMILES* $\rightarrow$ `Mol`. Where `Mol` is a molecule object from the RDKit library, more specifically, the `Chem` module. Then we're going to extract the features we want, and creates a new dataset from that.

An other steps of data cleaning and preprocessing consists in handling `NaN` and other invalid values. Here, such values will be handled by removing the corresponding rows.

From this, we're going to split the dataset into a training and a testing set.

A 70-30 split was used, with 70% of the data being used for training and 30% for testing. Removing the invalid data, we obtain 14030 samples in total. From which 9821 samples are used for training and 4209 for testing.

1.3 Presentation of the Models

In this project, in addition to the baseline model, three models were implemented to predict the pIC50 and logP values.

Namely:

- A random forest
- A KNN (K-Nearest Neighbors) model
- And a linear regression

2 Baseline Algorithm

2.1 Description

The baseline algorithm serves as a simple benchmark against which we can compare more sophisticated models. In our case, we use a mean prediction baseline.

2.2 Mathematical Formulation

For a given target variable y with n training samples, the baseline prediction $\hat{y}_{\text{baseline}}$ for any new input is:

$$\hat{y}_{\text{baseline}} = \frac{1}{n} \sum_{i=1}^{n} y_i$$

where y_i are the target values in the training set.

2.3 Implementation

In our project, we implemented the baseline algorithm as follows:

1. Calculate the mean values of pIC50 and logP from the training set:

$$\text{mean\_pic50} = \frac{1}{n} \sum_{i=1}^{n} \text{pIC50}_i$$

$$\text{mean\_logP} = \frac{1}{n} \sum_{i=1}^{n} \text{logP}_i$$

2. For each instance in the test set, predict these mean values:

$$\hat{y}_{\text{pic50}} = \text{mean\_pic50}$$

$$\hat{y}_{\text{logP}} = \text{mean\_logP}$$

2.4 Evaluation

We evaluate the baseline algorithm using Mean Squared Error (MSE):

$$\text{MSE}_{\text{baseline}} = \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_{\text{baseline}})^2$$

where m is the number of samples in the test set, y_i are the true values, and $\hat{y}_{\text{baseline}}$ is the baseline prediction (which is constant for all samples).

Moreover we can plot the predictions given the targets values and we have the following plots :

Figure 1: Baseline algorithm predictions given the target values

We see that this is realy not accurate since the predictions are only the mean.

3 Random Forest

3.1 Introduction

In this project, we use Random Forest to predict two molecular properties: pIC50 and logP. Random Forest is an ensemble learning method that operates by constructing multiple decision trees during training and outputting the mean prediction of the individual trees for regression tasks.

3.2 How Random Forest Works

3.2.1 Ensemble of Decision Trees

Random Forest is composed of multiple decision trees. If we denote the number of trees as B, we can represent the Random Forest model as:

$$\{T_1, T_2, \ldots, T_B\}$$

Each tree T_i is trained independently on a bootstrap sample of the original training data.

3.2.2 Bootstrap Aggregating (Bagging)

For each tree, a bootstrap sample is created by randomly sampling N instances from the training data with replacement, where N is the size of the training set. This process is called bootstrap aggregating or bagging. It introduces randomness and helps in reducing overfitting.

3.3 Random Feature Selection

At each node of a decision tree, instead of considering all features for splitting, Random Forest selects a random subset of features. If m is the number of features considered at each split, typically $m \approx \sqrt{p}$ for regression, where p is the total number of features.

3.3.1 Tree Growing

Each tree is grown to its maximum depth without pruning. At each node:

1. Randomly select m features.
2. Find the best split point among the m features.
3. Split the node into two child nodes.

3.3.2 Prediction

For a new input x, each tree makes a prediction. For regression, the final prediction is the average of all tree predictions:

$$f(x) = \frac{1}{B} \sum_{i=1}^{B} T_i(x)$$

where $T_i(x)$ is the prediction of the i-th tree.

3.4 Implementation Details

In our project, we used scikit-learn's RandomForestRegressor with the following key steps:

1. **Feature Engineering:** We extracted molecular descriptors using RDKit, including MolWt, NumBonds, NumC, NumO, NumN, NumHDonors, NumHAcceptors, NumAromaticRings, NumSaturatedRings, NumAliphaticRings, and NumRotatableBonds.
2. **Data Splitting:** We split the data into training (80%) and test (20%) sets.

3. **Hyperparameter Tuning:** We used GridSearchCV to find the best hyperparameters for both pIC50 and logP prediction. The hyperparameters tuned were:
   - n_estimators: [100, 200]
   - max_depth: [5, 10, 20, 30]
   - min_samples_split: [2, 5, 10]
4. **Model Training:** We trained separate Random Forest models for pIC50 and logP using the best hyperparameters found.
5. **Model Evaluation:** We evaluated the models using Mean Squared Error (MSE) on the test set and compared them to a baseline model (which predicts the mean of the training set).

3.5 How GridSearchCV Works

GridSearchCV is a technique used for hyperparameter optimization. It works by performing an exhaustive search over specified parameter values for an estimator. This method is particularly useful when we want to find the best combination of hyperparameters for our model.

GridSearchCV operates as follows:

1. **Parameter Grid Definition:** We define a grid of hyperparameter values to search over. For our Random Forest model, we defined:

```
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [5, 10, 20, 30],
    'min_samples_split': [2, 5, 10]
}
```

2. **Cross-Validation:** GridSearchCV uses k-fold cross-validation to evaluate each combination of parameters. In our case, we used 5-fold cross-validation.
3. **Exhaustive Search:** It tries all possible combinations of parameter values specified in the grid.
4. **Model Evaluation:** For each combination, it trains the model and evaluates its performance using the specified scoring metric (default is accuracy for classification and R-squared for regression).
5. **Best Model Selection:** After trying all combinations, it selects the best performing model based on the mean cross-validated score.

3.5.1 Mathematical Formulation

Let $\mathcal{M}$ be our model (Random Forest in this case), $\mathcal{D}$ be our dataset, and $\Theta = \{\theta_1, \theta_2, ..., \theta_n\}$ be the set of hyperparameters we want to tune.

For each combination of hyperparameters $\theta \in \Theta$, GridSearchCV computes:

$$\text{Score}(\theta) = \frac{1}{k} \sum_{i=1}^{k} \text{Score}_i(\mathcal{M}_\theta, \mathcal{D}_{\text{train}_i}, \mathcal{D}_{\text{val}_i})$$

where k is the number of folds in cross-validation, $\mathcal{M}_\theta$ is the model trained with hyperparameters θ, $\mathcal{D}_{\text{train}_i}$ and $\mathcal{D}_{\text{val}_i}$ are the training and validation sets for the i-th fold respectively.

The best hyperparameters θ^* are then selected as:

$$\theta^* = argmax_{\theta \in \Theta} \text{Score}(\theta)$$

3.5.2 Implementation in Our Project

We implemented GridSearchCV for both pIC50 and logP prediction:

```
# For pIC50
regressor_pic50 = RandomForestRegressor()
grid_pic50 = GridSearchCV(regressor_pic50, param_grid, cv=5)
grid_pic50.fit(X_train, y_pIC50_train
# For logP
regressor_logP = RandomForestRegressor()
grid_logP = GridSearchCV(regressor_logP, param_grid, cv=5)
grid_logP.fit(X_train, y_logP_train)
```

The best parameters found by GridSearchCV were then used to train our final Random Forest models.

3.5.3 Advantages of GridSearchCV

- **Systematic Search:** It ensures that we don't miss any combination of parameters.
- **Cross-Validation:** It uses cross-validation to provide a more robust estimate of model performance.
- **Automation:** It automates the process of hyperparameter tuning, saving time and reducing human error.

3.5.4 Limitations

- **Computational Intensity:** For large parameter grids or datasets, it can be computationally expensive.
- **Discretization:** It only searches through a discrete set of parameter values, potentially missing optimal values between the specified points.

3.6 Mathematical Formulation of MSE

The Mean Squared Error (MSE) is calculated as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

where n is the number of samples, y_i is the true value, and $\hat{y}_i$ is the predicted value.

3.7 Feature Importance

Random Forest provides a measure of feature importance based on how much each feature contributes to decreasing the weighted impurity across all trees. For a feature j, its importance is calculated as:

$$Imp(j) = \frac{1}{B}\sum_{i=1}^{B}\sum_{k \in N_j^i} p_k \Delta I(s_{k,j})$$

where N_j^i is the set of nodes in tree i where feature j is used for splitting, p_k is the proportion of samples reaching node k, and $\Delta I(s_{k,j})$ is the impurity decrease from splitting on feature j at node k.

Here is a representation of features importance for the pIC50 target :

Figure 2: Features Importances for the pIC50 target

We clearly see that "*num_n*" is the most important feature by far, meaning it has a big predictive impact on the pIC50 target.

3.8 Conclusion

Random Forest proved to be effective for predicting both pIC50 and logP, outperforming the baseline model. Its ability to handle non-linear relationships and provide feature importance makes it particularly suitable for molecular property prediction tasks.

3.9 Comparison of Baseline and Random Forest

To demonstrate the effectiveness of the Random Forest model, we compare its performance against the baseline algorithm:

1. **Mean Squared Error (MSE):** We compare the MSE of both models on the test set. A lower MSE for Random Forest indicates better predictive performance.

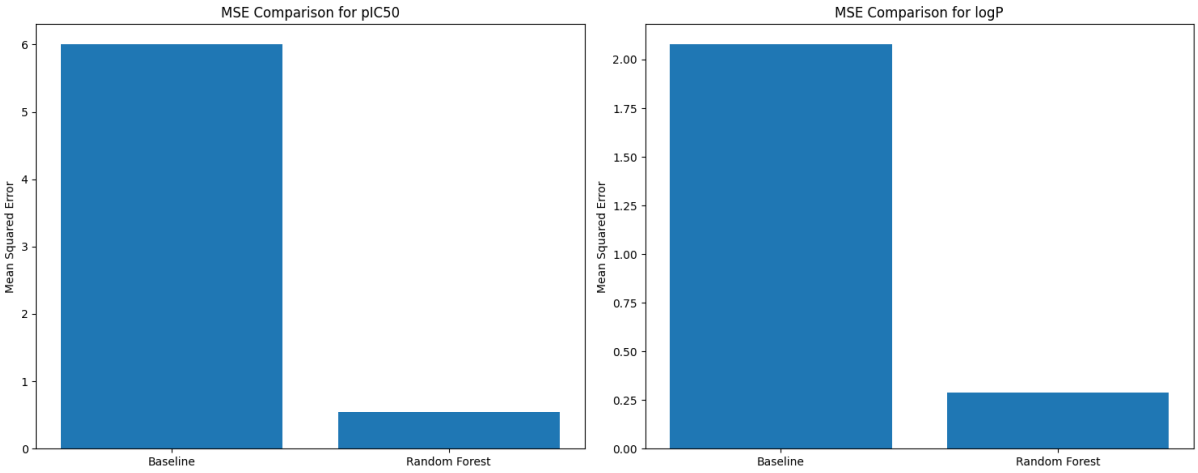

Figure 3: Mean squared error comparison for each target

2. **Visualization:** We create scatter plots of actual vs. predicted values for both models. The Random Forest predictions should show a tighter clustering around the diagonal line compared to the baseline predictions.

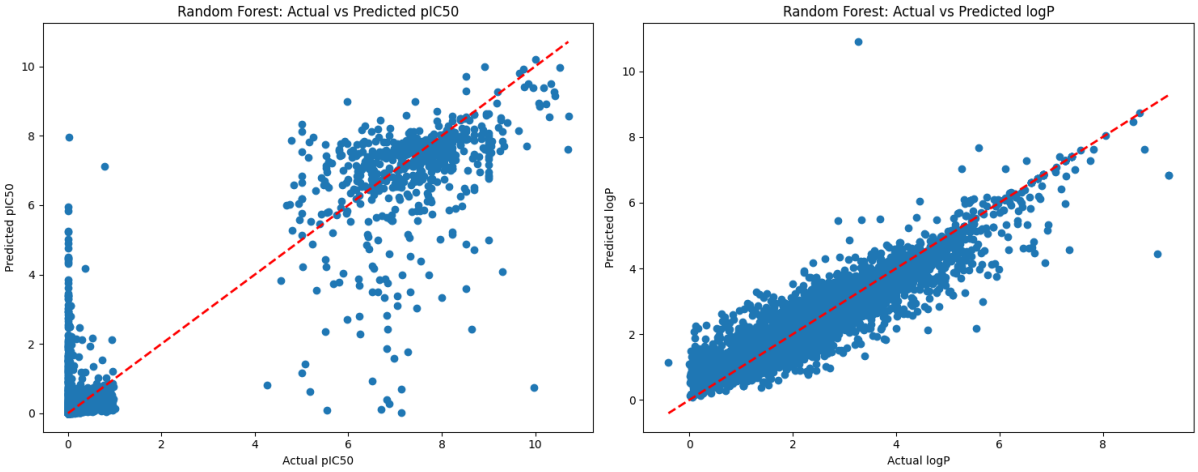

Figure 4: Scatter plot of random forest predictions

3. **Error Distribution:** We analyze box plots of the prediction errors for both models. The Random Forest errors have a smaller range and fewer outliers compared to the baseline errors.

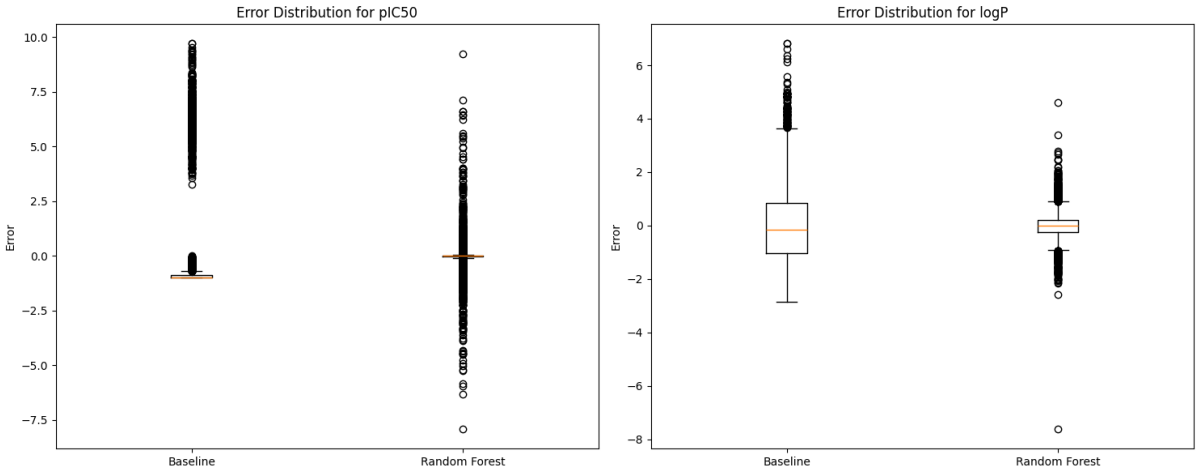

Figure 5: Error Distribution for both targets

This comparison allows us to quantify the improvement gained by using Random Forest over a simple mean prediction, justifying the use of a more complex model for our molecular property prediction task.

3.10 Conclusion

By comparing Random Forest to a baseline algorithm, we can demonstrate its superior performance in predicting both pIC50 and logP. The ability of Random Forest to capture complex, non-linear relationships in the data allows it to significantly outperform the simple mean prediction baseline, making it a suitable choice for molecular property prediction tasks.

4 KNN (K-Nearest Neighbors)

The K-Nearest Neighbors (KNN) algorithm is a popular non-parametric method used in various data mining and machine learning tasks, particularly for classification and regression.
It's a type of instance-based learning or non-generalizing learning: it does not construct an internal model but makes decisions based on the entire dataset. For regression tasks, KNN predicts the output value for a new data point by averaging the values of its K-nearest neighbors in the training set.

As previously stated, in this project, we will focus on using the KNN model for regression purposes to predict two critical pharmacological metrics: pIC50 and logP.

Use Cases in Pharmacology and Drug Discovery

- pIC50 Prediction: pIC50 is a measure used in pharmacology to assess the potency of a compound in inhibiting a specific biological target or enzyme. It is the negative logarithm of the IC50 value, which represents the concentration of a compound where 50% inhibition of its target is observed.
- logP Prediction: logP is a measure of a molecule's hydrophobicity, indicating how it interacts with polar (water-like) and nonpolar (oily) environments. It is a crucial factor in drug design, influencing a drug's absorption, distribution, metabolism, and excretion (ADME) properties.

Algorithm Overview:

- Distance Metric: To determine the K nearest neighbors, a distance metric (typically Euclidean distance) is used. The distance between a new data point $\mathbf{x}$ and each point $\mathbf{x}_i$ in the training set is calculated.

$$d(\mathbf{x}, \mathbf{x}_i) = \sqrt{\sum_{j=1}^{d} (x_j - x_{ij})^2}$$

- Identifying Neighbors: For a given test instance, the algorithm identifies the K training instances that are closest in terms of the chosen distance metric.
- Averaging for Prediction: The output value for the test instance is predicted by averaging the output values of these K nearest neighbors. For regression, the predicted value $\hat{y}$ is computed as:

$$\hat{y} = \frac{1}{K} \sum_{i \in \mathcal{N}_K} y_i$$

where $\mathcal{N}_K$ is the set of the K nearest neighbors, and y_i is the target value of the i-th neighbor.

Advantages of KNN for Regression:

- Flexibility: It can handle multi-modal data (data with multiple distributions).
- No Training Phase: KNN is a lazy learner, meaning it has no explicit training phase and uses all data points for making predictions.

Challenges and Considerations:

- Choice of K: The number of neighbors K significantly affects the performance of the model. A small K can lead to a model sensitive to noise, while a large K can smooth out important patterns.
- Computational Cost: The prediction phase can be computationally expensive, especially for large datasets, as it requires calculating the distance to all training samples.
- Curse of Dimensionality: The effectiveness of KNN decreases as the number of dimensions increases, making it less suitable for high-dimensional datasets.

4.1 Results

In evaluating the performance of the K-Nearest Neighbour (K-NN) model for predicting pIC50 and logP values, several key metrics were analyzed. The K-NN model parameters were optimized for the best performance and then compared against a baseline model that used the mean values of the training data for predictions.

4.1.1 K-Nearest Neighbour Performance

The K-NN model was optimized with the following parameters for both pIC50 and logP predictions:

- **Number of Neighbors (n_neighbors)**: 4
- **Distance Metric (p)**: 2 (Euclidean distance)

Wich gave the following results:

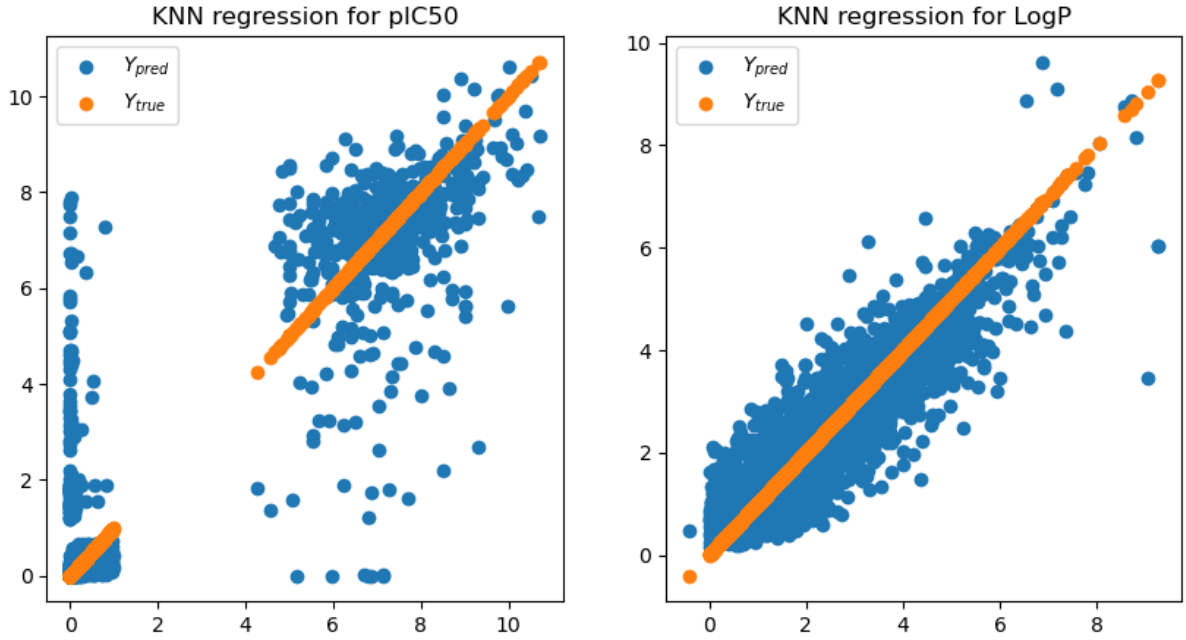

Figure 6: Scatter plot of KNN predictions

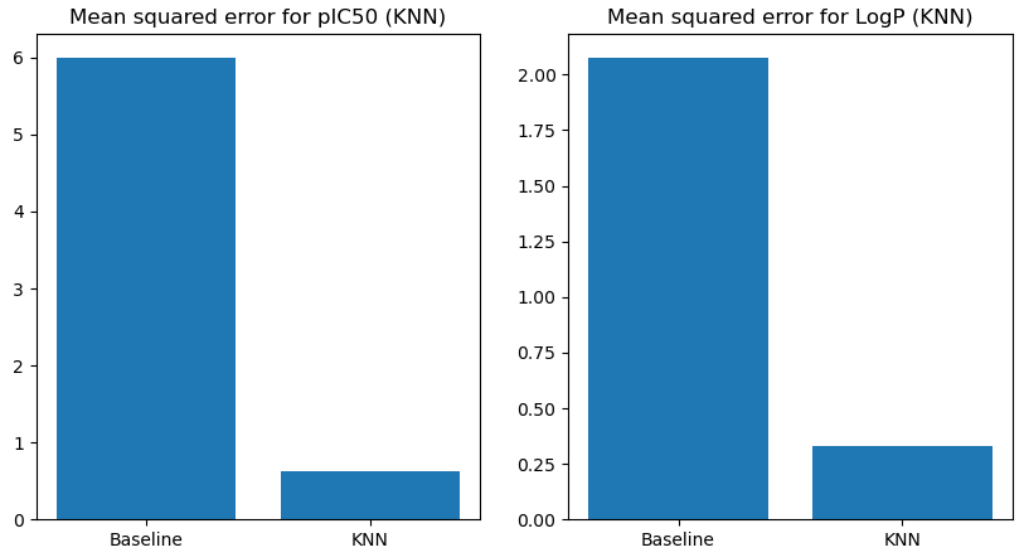

Figure 7: Comparison of MSE – Baseline VS KNN

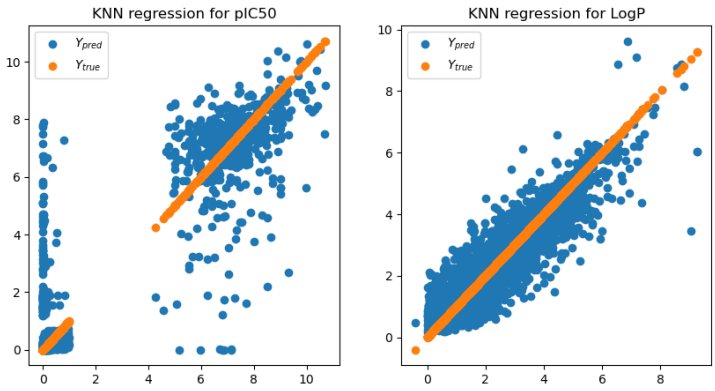

Figure 8: Scatter plot of KNN predictions

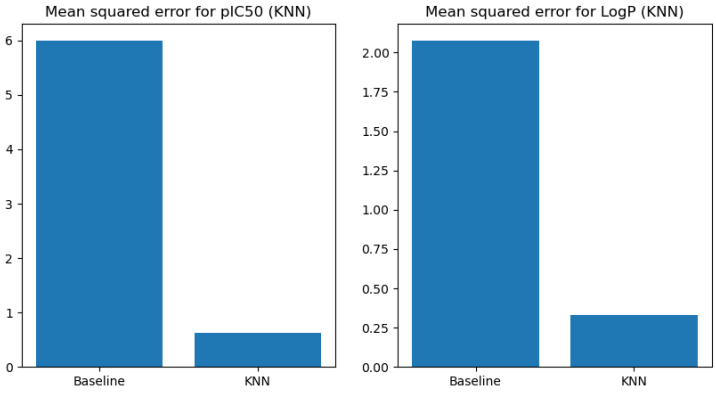

Figure 9: Comparison of MSE – Baseline VS KNN

The mean squared error (MSE) was used to evaluate the performance of the K-NN model on the test set. The results were as follows:

- **MSE for pIC50 (K-NN)**: 0.6204693303967199
- **MSE for logP (K-NN)**: 0.32986336327492954

The R-squared (R2) score was also used to evaluate the performance of the K-NN model. The results were as follows:

- **R2 score for pIC50 (K-NN)**: 0.90
- **R2 score for logP (K-NN)**: 0.84

The R2 score, being a statiscal measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model. i.e. How well the model predicts the outcome of the dependent variable.

On a maximum of 1, the R2 score indicates that the model predicts 90% of the relationship between the dependent variable and the independent variable for pIC50 and 84% for logP.

4.1.2 Comparative Analysis

When comparing the performance of the K-NN model to the baseline model, it is evident that the K-NN model provides a more accurate prediction for both pIC50 and logP values. The baseline model, serving as a simplistic benchmark, obtained the following MSE:

- **MSE for pIC50**: 6.955
- **MSE for logP**: 2.02

Which is approximately 10 times higher than the MSE values obtained by the K-NN model. This significant difference highlights the effectiveness of the K-NN algorithm in capturing the underlying patterns in the data and making accurate predictions.

4.1.3 Conclusion

The results indicate that while the baseline model provides a straightforward benchmark, the K-NN model significantly enhances the prediction accuracy for both pIC50 and logP values. This improvement highlights the effectiveness of the K-NN algorithm in handling molecular data and predicting important pharmacological properties. The optimized parameters of $n\_neighbors = 4$ and $p = 2$ demonstrate that the K-NN model can leverage local data structures to provide more reliable predictions, thus supporting its application in drug discovery and pharmacology.

5 Linear Regression

5.1 Introduction

In this section, we explore the use of linear regression as a baseline model to predict two critical pharmacological properties: pIC50 and logP. Given the complexity of molecular interactions, a linear relationship between the features and the targets is unlikely, but starting with a simple model helps to establish a benchmark for performance.

5.2 Model Description

Linear regression aims to establish a relationship between the dependent variable (target) and one or more independent variables (features) by fitting a linear equation to the observed data. In our case, we seek to predict two targets: pIC50 and logP, using a set of molecular descriptors extracted from the dataset.

Mathematically, the relationship can be expressed as:

$$Y = X\beta$$

Where: - Y is the matrix of target values (pIC50 and logP), - X is the matrix of feature values (molecular descriptors), - β is the matrix of coefficients that we need to estimate.

For a given dataset with n samples and d features, the coefficient matrix β has dimensions $(d + 1) \times h$, where h is the number of targets (2 in our case).

5.3 Methodology

1. **Training and Validation**:
   - The data was split into training and test sets.
   - We used K-Fold Cross-Validation with 5 splits to optimize the hyperparameters and evaluate the model's performance. This helps ensure that our model generalizes well to unseen data by training on different subsets of the data and validating on the remaining parts.
2. **Model Training**:
   - Two linear regression models were trained separately for pIC50 and logP.
   - The models were fitted on the training data to learn the coefficients β.
3. **Evaluation Metrics**:
   - The Mean Squared Error (MSE) was used to evaluate the model's performance. MSE measures the average squared difference between the actual and predicted values, providing a sense of how close the predictions are to the actual targets.

The code snippet below outlines the implementation:

```
model_pic50 = LinearRegression()
model_logp = LinearRegression()

kfold = KFold(n_splits=5, shuffle=True, random_state=69)

# Function to perform cross-validation and return average scores
def cross_val_score_avg(model, X, y, kf):
    scores = []
    for train_idx, val_idx in kf.split(X):
        X_train_fold, X_val_fold = X[train_idx], X[val_idx]
        y_train_fold, y_val_fold = y.iloc[train_idx], y.iloc[val_idx]

        model.fit(X_train_fold, y_train_fold)
        pred = model.predict(X_val_fold)

        scores.append(mean_squared_error(y_val_fold, pred))
    return np.mean(scores)

# Function to train a model and return the test set MSE
def train_and_evaluate_model(model, X_train, y_train, X_test, y_test):
    model.fit(X_train, y_train)
    pred_test = model.predict(X_test)
    mse_test = mean_squared_error(y_test, pred_test)
    return mse_test, pred_test

mse_pic50_val = cross_val_score_avg(model_pic50, X_train, y_pIC50_train, kfold)
mse_logp_val = cross_val_score_avg(model_logp, X_train, y_logP_train, kfold)

print(f"MSE of validation set for the pIC50 model: {mse_pic50_val}")
print(f"MSE of validation set for the logP model: {mse_logp_val}")

mse_pic50_test, pred_pic50 = train_and_evaluate_model(model_pic50, X_train, y_pIC50_train, X_test, y
mse_logp_test, pred_logp = train_and_evaluate_model(model_logp, X_train, y_logP_train, X_test, y_log

print(f"MSE of test set for the pIC50 model: {mse_pic50_test}")
print(f"MSE of test set for the logP model: {mse_logp_test}")
```

5.4 Results and Discussion

The linear regression model served as a straightforward approach to establish a performance baseline. The cross-validation results provided insight into how well the model might generalize to unseen data. The final MSE values on the test set indicated the baseline performance for both targets.

- **pIC50 Model**:
  - Validation MSE: 2.914361592443585
  - Test MSE: 2.7677131085189135
- **logP Model**:
  - Validation MSE: 0.5122910557928586
  - Test MSE: 0.4929892547286209

We can see in the figure below that the pIC_{50} is not well predicted, with predicted values far from the actual values. On the other hand, the $logP$ is predicted much better, as we can see in the figure.

These results demonstrate that while linear regression can provide an initial understanding of the data, more complex models may be necessary to capture the non-linear relationships inherent in molecular data. Future steps will involve experimenting with more sophisticated models and feature extraction techniques to improve predictive performance.

Comparison of pIC_{50} and $logP$ predictions: