Comprehensive Analysis of Logistic Regression and K-Nearest Neighbors for QSAR Data

Elena Manoli

***Abstract*—This document conducts a comprehensive analysis of logistic regression modeling for QSAR data. It encompasses data processing, exploratory data analysis, and the application of logistic regression, and K-Nearest Neighbors (KNN). The data processing phase involves loading, inspecting, and cleaning the dataset, while exploratory data analysis provides insights into feature distributions and interrelationships. Methodology details the implementation of logistic regression, and KNN, with hyperparameter tuning and model analysis metrics. Findings include insights into each model's performance, strengths, and limitations.**

I. Introduction

Quantitative Structure-Activity Relationship (QSAR) modeling plays a pivotal role in predicting the biological activity of chemical compounds, particularly in understanding the biodegradability of chemicals [1]. The accurate prediction of biodegradability is crucial in preventing the harmful accumulation and environmental dispersion of substances. QSAR, a chemoinformatics modeling technique, bridges the gap between chemical structures and their biological or chemical activities. This study focuses on logistic regression and K-Nearest Neighbors (KNN) as robust tools for QSAR data analysis. Logistic regression offers interpretability and simplicity, making it well-suited for modeling the probability of biodegradability. On the other hand, KNN provides the flexibility to capture non-linear relationships inherent in chemical structures, contributing to a more comprehensive understanding of their biological effects. The importance of choosing an appropriate model cannot be overstated, as accurate predictions are essential for informed decision-making in environmental science.

II. Data Processing

A. Loading and Initial Inspection

The initial loading of data utilized the `scipy.io.loadmat` function, extracting information from the provided Matlab data files. Subsequently, the data underwent conversion into a Pandas DataFrame to facilitate advanced analysis and manipulation. A thorough inspection of the loaded data, utilizing the `info()` and `describe()` functions, revealed its structure. Comprising a total of 42 columns, the dataset contains features across the first 41 columns, with the final column dedicated to labels.

E. Manoli is with the Department of Automatic Control and Systems Engineering, The University of Sheffield. Email: emanoli1@sheffield.ac.uk

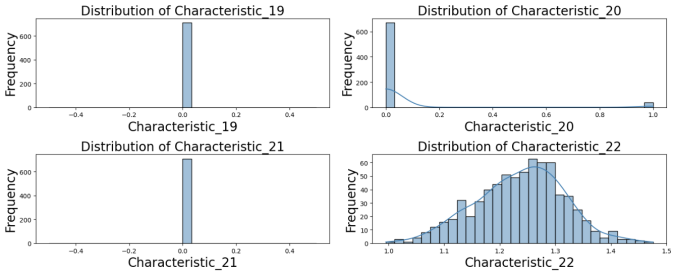

Fig. 1. Histograms depicting the distribution of four features

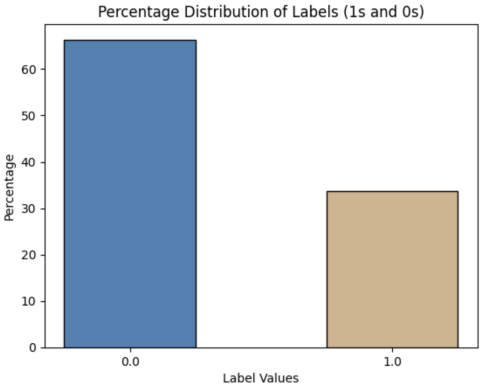

Fig. 2. Percentage Distribution of Labels (1s and 0s).

B. Exploratory Data Analysis

To gain deeper insights into the dataset's features, histograms were generated. These visual representations showcased the frequency distribution of each feature, offering a detailed understanding of the underlying data structure. 1 illustrates four out of the 41 histograms, providing a snapshot of the feature distribution.

In the process of examining these histograms, it became apparent that certain values in the dataset significantly diverged from the majority of the data. These extreme values, commonly referred to as outliers, are points that lie far from the central tendency of the distribution. Outliers can introduce distortion to statistical measures and visualizations, potentially misrepresenting the overall characteristics of the dataset.

In Figure 1, the graph titled "Distribution of feature_20" highlights instances where certain features exhibit values that are notably distant from the majority of observations. The presence of these outliers can influence the interpretation of the feature distribution, affecting both the visual representation and statistical summary.

In addition to the histograms, a bar plot as shown in Figure

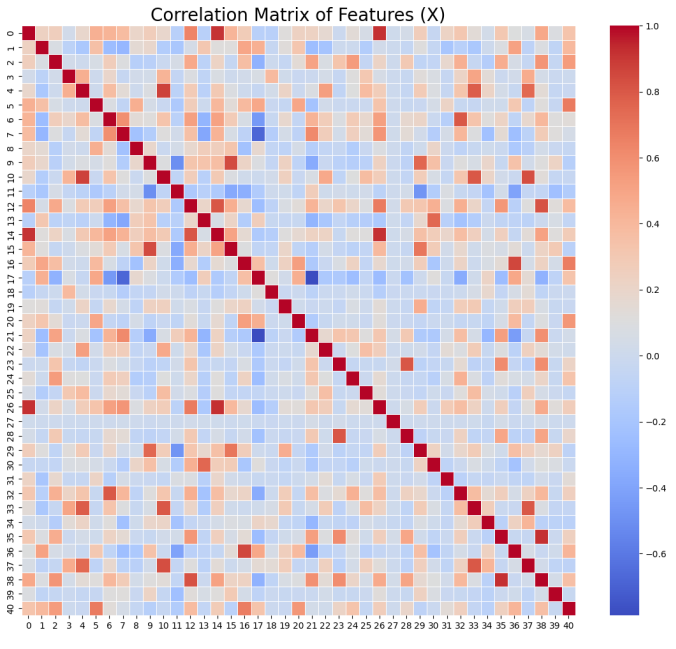

Fig. 3. Correlation Matrix of Features (X).

2 was crafted to visually present the distribution of label values. This plot provides a clear depiction of the percentage distribution of label values (1s and 0s) within the labeled data. The x-axis denotes the label values, while the y-axis signifies the corresponding percentages. Approximately 65% of the data points are labeled as 0, while the remaining 35% are labeled as 1. This distribution provides insights into the imbalance between the two classes, with the majority of instances belonging to the 0 class.

To comprehend the interrelationships between different features, a correlation matrix was constructed for the dataset's feature variables. The correlation matrix provides a numerical representation of the degree to which each feature correlates with every other feature. Figure 3 illustrates the correlation matrix, with each cell representing the correlation coefficient between two features. The color intensity and the numerical values in each cell convey the strength and direction of the correlation.

Highly correlated features may have redundant information, and their inclusion in certain machine learning models could lead to overfitting. Conversely, uncorrelated features may provide complementary information, enhancing the model's performance. This visualization aids in identifying potential patterns and relationships within the feature set, guiding subsequent feature selection.

C. Data Cleaning and Preprocessing

To enhance data integrity, the dataset underwent several critical preprocessing steps. Firstly, duplicate rows were identified and removed to mitigate potential biases arising from redundant information. Following this, any rows containing missing values were eliminated from the dataset. Additionally, any remaining missing values in the features were carefully filled with zeros.

Considering the potential impact of outliers on the reliability of our analyses, a decision was made to employ outlier removal techniques. To identify outliers, Z-scores were calculated for each data point using the formula $Z = \frac{(X-\mu)}{\sigma}$, where X is the data point, μ is the mean, and σ is the standard deviation. A threshold of 3 standard deviations was set, classifying data points beyond this range as outliers. Rows containing such outliers were then removed.

By removing the outliers, the dataset transformed from a shape of (1052, 42) to (710, 42).This process ensures that statistical measures and visualizations are based on a more representative dataset, reducing the impact of extreme values. The Z-score threshold can be adjusted based on dataset characteristics and analysis goals.

D. Splitting the data

The dataset was divided into training and test sets using the `train_test_split` function from the scikit-learn library. The split was performed with a test size of 20%, ensuring that 80% of the data was used for training the model, while the remaining 20% was reserved for evaluating its performance. The use of a random state parameter (set to 42) ensures reproducibility in the split, enabling consistent results across different runs. This partitioning strategy is essential for assessing the model's generalization on unseen data and helps prevent overfitting by providing an independent dataset for evaluation.

E. Data Standardization

Different scales among the features in a dataset can significantly impact the modeling process, introducing bias into the outcome of predictions in terms of misclassification error and accuracy rates. To address this issue and ensure fair and unbiased model training, the StandardScaler is applied to standardize the dataset. This procedure adjusts each variable to have a mean of 0 and a standard deviation of 1. The goal is to normalize the impact of each variable, preventing any individual feature from unduly affecting the learning process solely based on its original scale. The standardization involved calculating the mean and standard deviation for each feature using the formula: $z = \frac{x - \text{mean}(x)}{\text{std}(x)}$.

For each individual feature represented by the variable x, the mean $\text{mean}(x)$ and $\text{std}(x)$ standard deviation of the feature were calculated. This formula was applied individually to each element within the dataset.

III. Methodology

A. Model Selection

Logistic regression was chosen for this binary classification task due to its simplicity, interpretability, and effectiveness in modeling the probability of a binary outcome. Logistic regression is well-suited for scenarios where the relationship between the features and the binary response variable is expected to be approximately linear. Additionally, it is less prone to overfitting, making it a suitable choice for datasets with a moderate number of features.

K-Nearest Neighbors (KNN) was selected as an alternative model for this binary classification task. It is known for its simplicity and flexibility, making it effective in capturing non-linear relationships in data. It classifies a data point based on the majority class of its k-nearest neighbors, where the distance metric plays a crucial role in defining proximity.

B. Implementation Details for Logistic Regression

The logistic regression algorithm was implemented from scratch in Python, encompassing the essential components for model training and prediction. The implementation includes the following key components:

1) **Sigmoid Function:** It converts the linear combination of input features into probabilities between 0 and 1, representing the likelihood of the binary outcome. It is defined as $\sigma(z) = \frac{1}{1+e^{-z}}$, where z is the linear combination of weights and features.
2) **Gradient Descent:** is the iterative optimization algorithm at the heart of logistic regression, driving the model to find optimal parameter values. The gradients of the cost function with respect to the model parameters (θ and b) guide the update process. For each iteration, the weights and bias are adjusted in the opposite direction of the gradients, moving the model towards the minimum of the cost function. The update rules are defined as: $\theta_j = \theta_j - \alpha\frac{\partial J}{\partial \theta_j}$ and $b = b - \alpha\frac{\partial J}{\partial b}$ where α is the learning rate, and $\frac{\partial J}{\partial \theta_j}$ and $\frac{\partial J}{\partial b}$ are the partial derivatives of the cost function with respect to the weights and bias, respectively.
3) **Regularization:** helps control the complexity of the model by penalizing large weights. The regularization parameter (λ) is a hyperparameter that can be tuned to balance model complexity and fit.

C. Implementation Details for K-Nearest Neighbors (KNN)

The KNN algorithm was implemented in Python, encompassing key components for model training and prediction:

1) **Euclidean Distance:** For each data point in the testing set, the algorithm calculates its distance from every point in the training set using the Euclidean distance formula: $\text{distance}(\text{point}, \text{data}) = \sqrt{\sum_{i=1}^{n}(\text{point}_i - \text{data}_i)^2}$
This quantifies the similarity or dissimilarity between data points.
2) **Number of Neighbors(K):** The optimal number of neighbors (K) was determined through experimentation and cross-validation. This parameter significantly impacts the model's sensitivity to noise and generalization to unseen data.

D. Hyperparameter Tuning:

The model's hyperparameters, including the learning rate and number of iterations were selected based on experimentation and validation performance. Hyperparameter tuning is a critical step in achieving optimal model performance and generalization to unseen data.

IV. Model Analysis

The following metrics where used to provide a comprehensive understanding of how well each model generalizes to unseen data and captures the complexities of the underlying classification task.

1) **Accuracy:** measures the overall correctness of predictions, representing the proportion of correctly classified instances out of the total.
2) **F1-score:** is a robust metric for evaluating the performances of classification models, and mathematically F1-score is the harmonic mean of precision and recall.
3) **ROC Curve:** graphically depicts the model's trade-off between true positive rate and false positive rate across different threshold values.
4) **Confusion Matrix:** breaks down predictions into true positives, true negatives, false positives, and false negatives, providing a detailed overview of the model's performance across different classes.

A. K-Nearest Neighbors (KNN)

KNN, known for its simplicity and flexibility, requires the selection of the optimal number of neighbors (k). To identify the most suitable k, an exhaustive search was conducted over a range from 1 to 30. To visualize the impact of different k values on the model's accuracy, a graph was plotted. The x-axis represents the values of k (number of neighbors), and the y-axis represents the corresponding accuracy achieved on the test data.

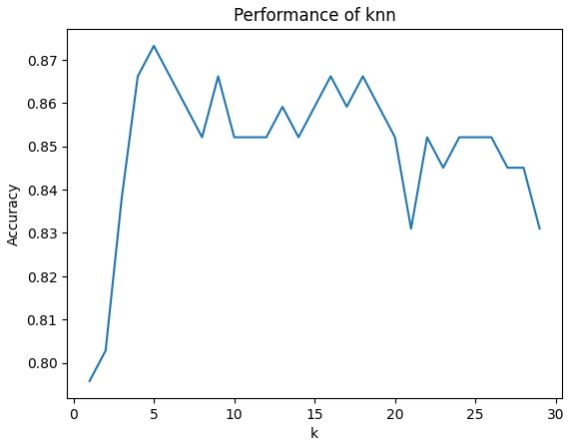

Fig. 4. Impact of k values on KNN Model Accuracy

As seen in Figure 4, it is evident that the accuracy varies with different k values. The point at which the accuracy is maximized represents the optimal k for this specific dataset is for k=5, which returns the following metrics.

- **Accuracy Score:** It achieved an accuracy score of 88.73% on the training data and 87.32% on the test data.
- **F1-Score:** 82.0%, **Precision:** 87.23%, **Recall:** 77.36

A lower accuracy on the test set compared to the training set could indicate challenges in generalizing well to new, unseen data. It also may be indicative of overfitting, where the model overly fits itself to the specific data of the training set. The

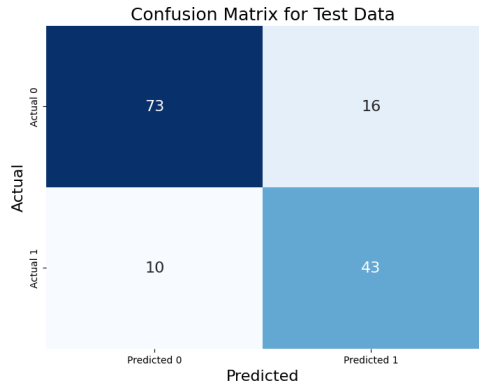

Fig. 5. Confusion Matrix of Logistic Regression for Testing data

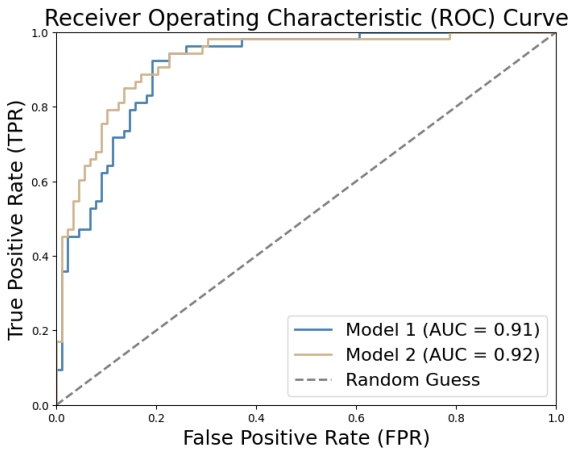

Fig. 6. ROC Curve Comparison: Logistic Regression vs. Regularized Logistic Regression

high precision score indicates that when the model predicts the positive class, it is accurate in the majority of cases. The moderate recall score suggests that the model captures a substantial portion of the true positive instances but may miss some positive cases. A high F1-score indicates a good balance between precision and recall, showcasing the model's ability to make accurate positive predictions while considering false positives and false negatives.

B. Logistic Regression without Regularization

- **Accuracy Score:** The logistic regression model without regularization achieved an accuracy score of 79.40% on the training data and 81.69% on the test data.
- **F1-Score:** 76.79%
- **Training data Confusion Matrix:** 175 instances were correctly predicted as class 1, 276 instances were correctly predicted as class 0, 79 instances were wrongly predicted as class 1 when they were actually class 0, 38 instances were wrongly predicted as class 0 when they were actually class 1.
- **Testing data Confusion Matrix:** As seen in Figure 5 43 instances were correctly predicted as class 1, 73 instances were correctly predicted as class 0, 16 instances were wrongly predicted as class 1 when they were actually class 0, 10 instances were wrongly predicted as class 0 when they were actually class 1.

For training data, the model seems to have a good number of correct predictions, but it also made errors in predicting both classes. Similarly, in the testing data, the model performed well in terms of correct predictions, but it still made some errors, particularly in misclassifying instances of class 1.

Upon reviewing the initial results, an exploration was undertaken to evaluate the influence of the regularization parameter on the Logistic Regression model. Subsequently, it was observed that the introduction of the regularization parameter led to notable changes in accuracy metrics. Specifically, the accuracy score for the training data increased from 79.40% to 80.99%, while the accuracy score for the test data saw a more substantial improvement from 81.69% to 85.21%. These adjustments underscore the positive effect of the regularization parameter on enhancing the model's accuracy on both the training and test datasets.

The ROC curve shown in Figure 6 compares two logistic regression models: one without regularization and the other with regularization. The regularized model consistently outperforms its non-regularized counterpart, exhibiting a steeper ascent towards the top-left corner, indicating enhanced sensitivity and specificity. This visual distinction suggests that regularization improves the model's ability to discriminate between positive and negative instances. The higher Area Under the Curve (AUC) for the regularized model, reinforces the positive impact of regularization on discrimination performance. The observed separation between the curves underscores regularization's efficacy in enhancing the overall predictive accuracy and reliability of the logistic regression model.

V. Conclusion and Recommendation

Logistic regression's advantages lie in its interpretability and simplicity, making it effective for scenarios with approximately linear relationships. However, it struggles to capture complex non-linear patterns and is sensitive to outliers. K-Nearest Neighbors (KNN) excels in capturing non-linear relationships and handling diverse data patterns. However, it comes with computational expenses, especially with large datasets, and is sensitive to irrelevant features, requiring careful tuning of hyperparameters like the number of neighbors (k).

In conclusion, the choice between KNN, Logistic Regression, and Regularized Logistic Regression depends on specific characteristics and goals. If interpretability is crucial and the relationship between features is expected to be linear, Logistic Regression might be preferred. Regularized Logistic Regression is recommended when mitigating overfitting is a priority, especially in high-dimensional spaces. KNN, with its simplicity and adaptability, is suitable for quick prototyping but may require careful tuning to avoid overfitting.

References

[1] K. Mansouri, T. Ringsted, D. Ballabio, R. Todeschini, and V. Consonni, "Quantitative structure–activity relationship models for ready biodegradability of chemicals," Journal of chemical information and modeling, vol. 53, no. 4, pp. 867–878, 2013.