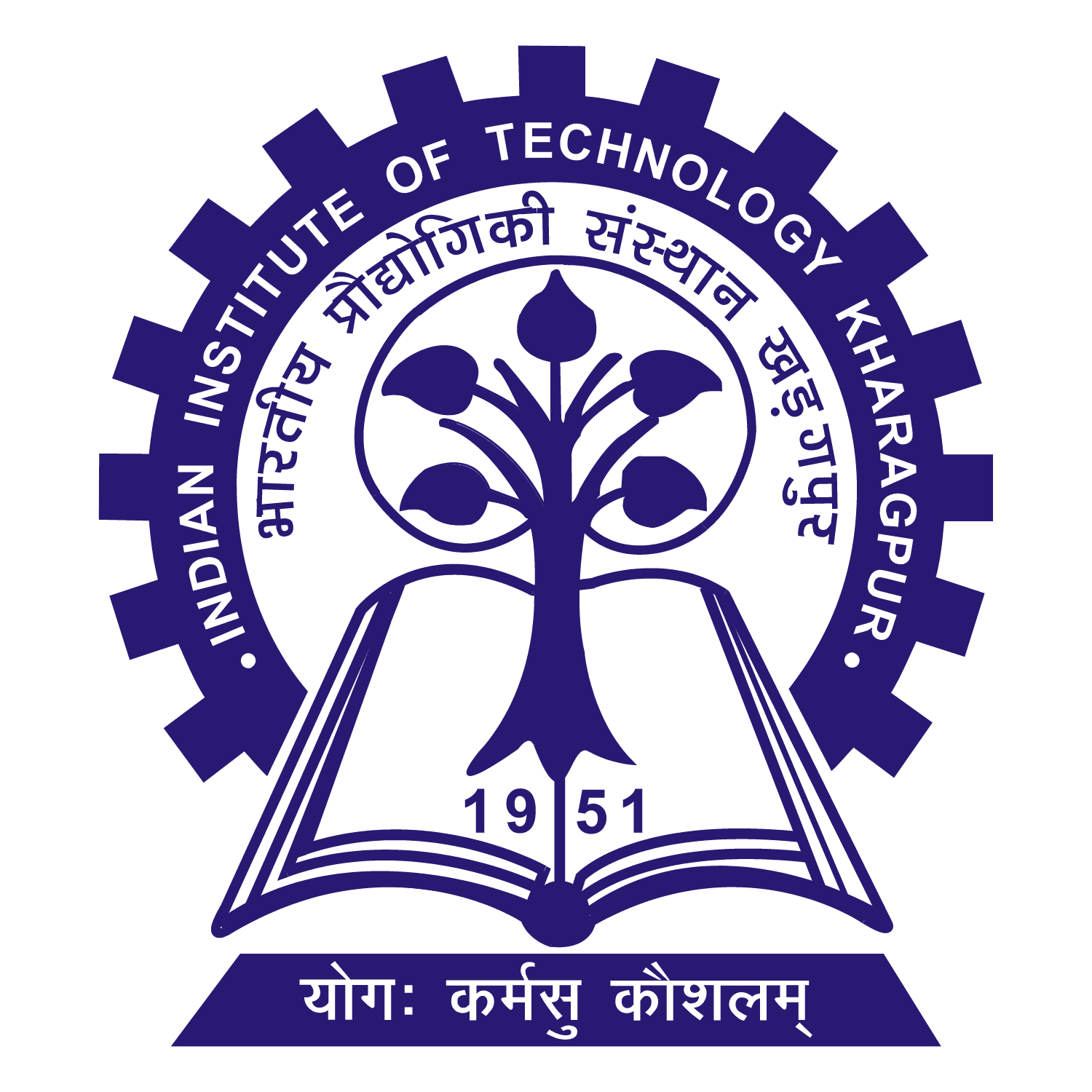
**Polydipsia Prediction using Gaussian Naive Bayes Classifier Learning Model**

**Course code: CS60050**

**Course Name: Machine Learning**



Group Number : 8

Project Code : PPNB

Prepared By:

Ravi Karthik (18EC3AI19)

Paneliya Yashkumar Shaileshbhai (22CS60R70)

Sayak Roy (22CS60R04)

**Problem statement:**

Dr. Strange’s Lab has decided to build a machine learning model for predicting polydipsia, a medical disease of excessive thirst, for women. The model will take the various diagnostic results (like hormone level, blood pressure etc.) and patient's conditions (like age, BMI, lineage factor) as input features and predict whether she has polydipsia. Moreover, they have decided to use a probabilistic machine learning model as they want the model to output a prediction probability (showing the model’s confidence on its prediction) rather than just 0 or 1. In particular, they have chosen Naive Bayes classifiers as they can add new features later without re-training the model for the existing features. Your task is to help Dr. Strange’s Lab to build the Naive Bayes classifier.

More precisely, your tasks are the following:

1. You will write a class to implement a Gaussian Naive Bayes classifier. The class should implement the following method:

a. Train: the method will take the train data as input and train the classifier on the data.

b. Predict: the method will take the test data as input and return the prediction probabilities on the data.

2. You will implement and try several feature transformation methods on the input features to improve the performance of the classifier.

3. Finally, you should generate results on the given data and compare those with the results obtained from the Gaussian Naive Bayes classifier of the scikit-learn package.

**Approach**:

**Data splitting and k-folds:**

The dataset was randomly partitioned into 5 parts and the model was trained five times using each set as the validation set in a round-robin manner.

**Algorithm**:

1. Created a class to describe our Gaussian Naive bayes classifier
2. Class has private variables like mean, standard deviation and prior probabilities of attributed
3. Train algorithm:
4. The first step in the method is to calculate the prior probabilities of the two classes. This is done by counting the number of instances of each class and dividing it by the total number of instances. The prior probabilities are stored in a dictionary self.prior.
5. The next step is to split the training data into two parts based on the class label. The class labels are stored in the 'Prediction' column of the dataframe. The two parts of the data are stored in two separate dataframes, class0\_data and class1\_data.
6. The 'Prediction' column is then dropped from both dataframes as it is not needed for further processing.
7. The final step is to calculate the mean and standard deviation of each feature for each class. This is done by looping over each column in the dataframes, calculating the mean and standard deviation of that column, and storing the results in dictionaries self.means and self.sd, respectively. The keys in these dictionaries correspond to the class labels, and the keys of each dictionary correspond to the feature names.

4. Predict/test algorithm:

1. The first step in the method is to extract the features and the target variable from the input data. The features are stored in a dataframe x\_test and the target variable is stored in a pandas series y\_test. The features to be used for prediction are specified in the column\_list list.
2. A list y\_pred is then initialized to store the predicted class labels for each instance.
3. The method then calculates the posterior probabilities of each class for each instance in the test data. This is done by looping over the instances in x\_test, and for each instance, calculating the product of the prior probabilities and the probabilities of each feature given the class. The probabilities are calculated using the Gaussian distribution and the mean and standard deviation values stored in the self.means and self.sd dictionaries, respectively. The probability of a feature given a class is calculated using a separate function find\_norm\_prob, the implementation of which is not shown in the code snippet.
4. The class label with the highest posterior probability is then chosen as the prediction for that instance. If the probability of class 0 is greater than that of class 1, a 0 is added to the y\_pred list, and if the probability of class 1 is greater than that of class 0, a 1 is added to the list.
5. Finally, the method returns the predicted class labels stored in y\_pred and the true class labels stored in y\_test.

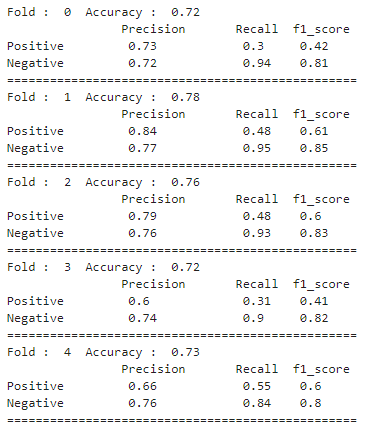
5. Score calculations:

1. We wrote our own function imitating the same behavior as library function to generate performance score.
2. Given the actual target values (y\_test) and predicted target values (y\_pred), it calculates the following evaluation metrics:
   1. True positive (TP)
   2. False positive (FP)
   3. True negative (TN)
   4. False negative (FN)
   5. Accuracy: (TP + TN) / (TP + TN + FP + FN)
   6. Precision (positive class): (TP) / (TP + FP)
   7. Recall (positive class): (TP) / (TP + FN)
   8. F1-Score (positive class): 2 / ((1/Precision) + (1/Recall))
   9. Precision (negative class): TN / (TN + FN)
   10. Recall (negative class): TN / (FP + TN)
   11. F1-Score (negative class): 2 / ((1/Precision) + (1/Recall))
3. It then stores these metrics in a dictionary and returns the report.

**Results:**

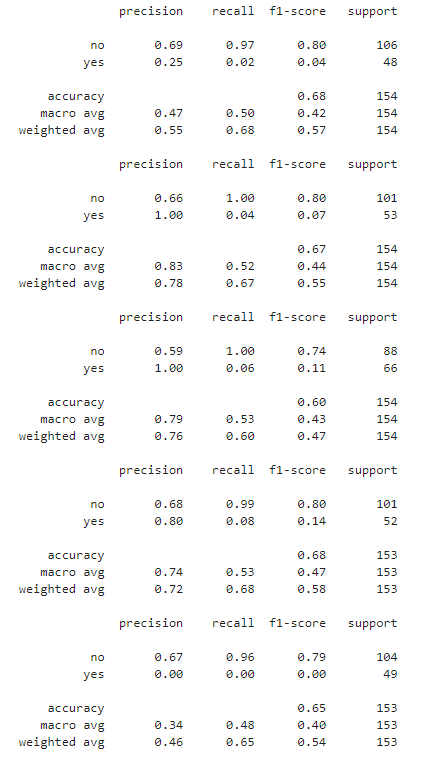
The results on the validation sets are as follows:

**Our model:**



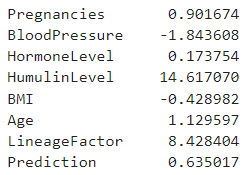
*Note: These scores are generated by the custom function* ***(test\_report(y\_pred,y\_test))*** *we wrote. The score matches with the library function classification\_report.*

The same model was also trained using the scikit-learn library using five-fold cross-validation. The accuracy of the folds, in this case, is given by:



**Skewness:**

The skewness of the original features is given by:

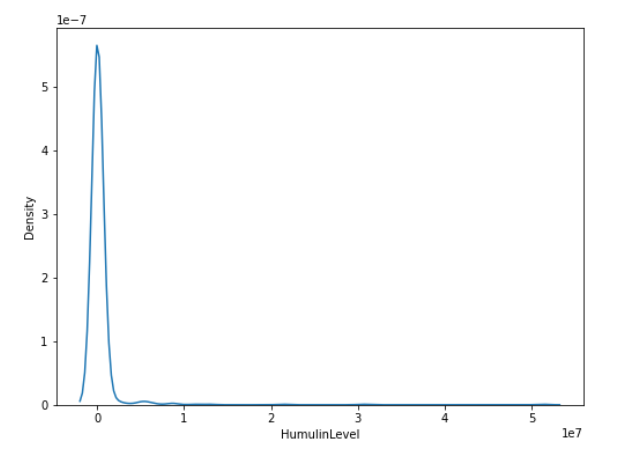


Here, we can see that only ‘HumulinLevel’ and ‘LineageFactor’ is showing large enough skewness. Hence, we apply feature transformation only to these features.

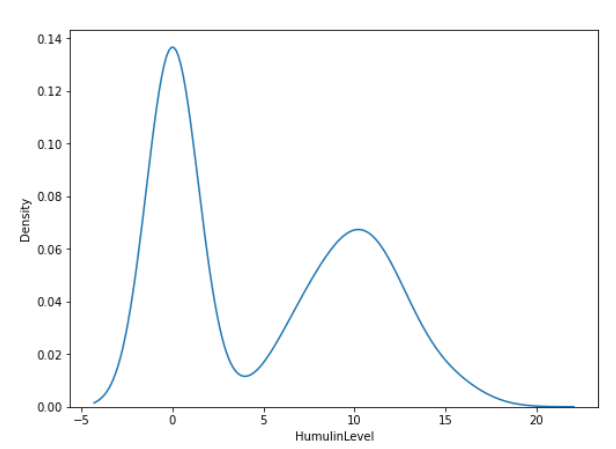
**Feature Transformation**

1. HumulinLevel:

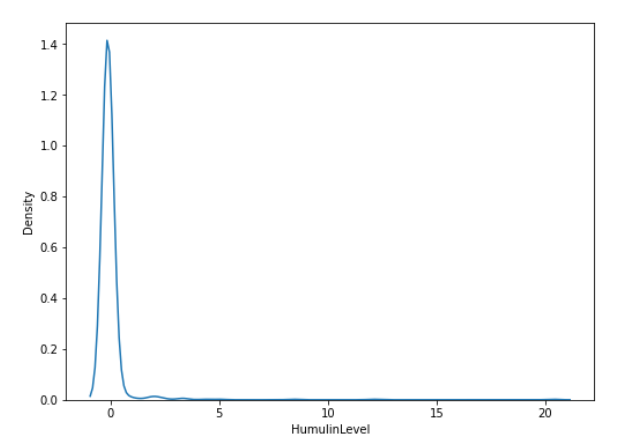
The kde-plot of ‘HumulinLevel’ is as shown:



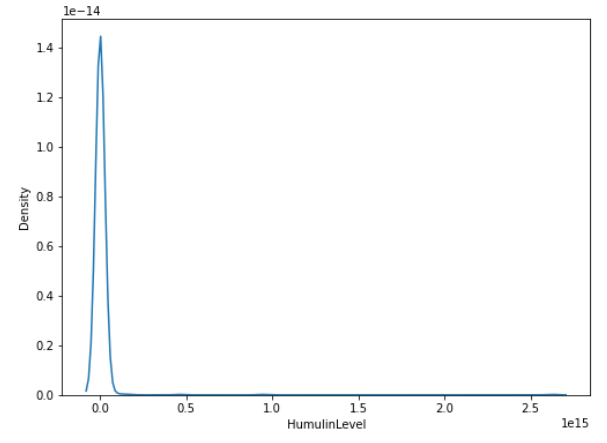
Applying log(1+x) transformation:



Applying z-norm transformation:



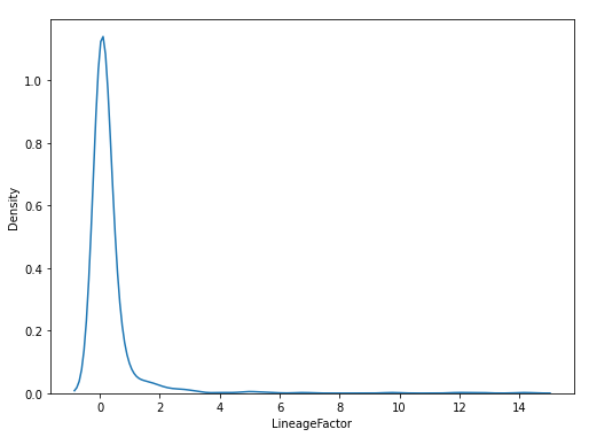
Applying quadratic transformation:



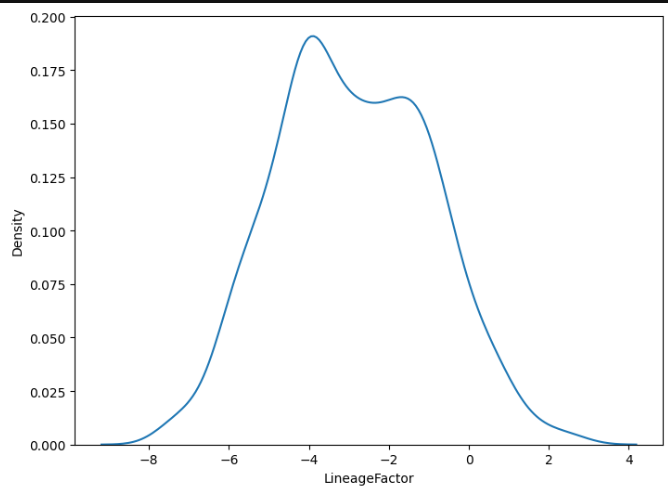
Among this, we can see that the log(1+x) transformation gives a plot closest to a bell-shaped curve. Hence log(1+x) was chosen as the function for transforming ‘HumulinLevel’

1. LineageFactor:

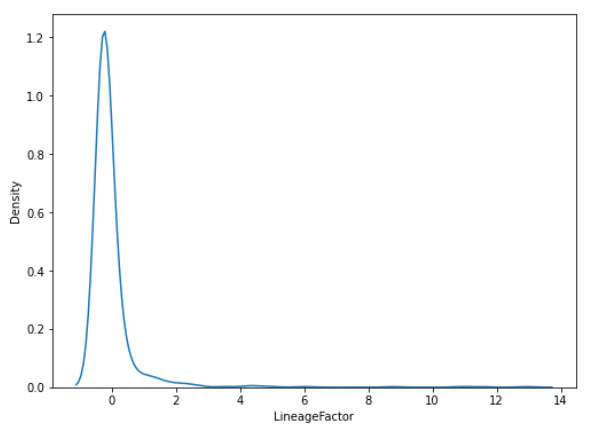
The kde-plot of ‘LineageFactor’ is as shown:



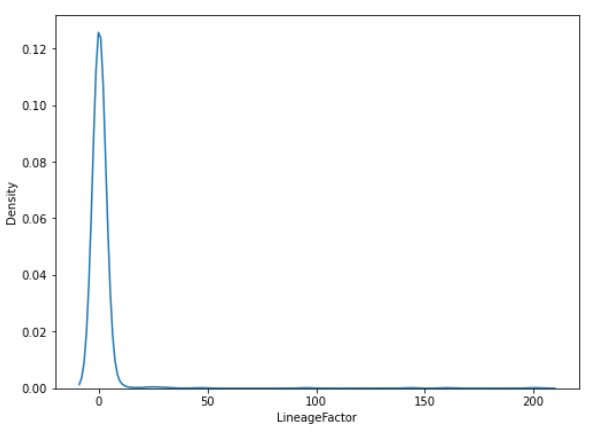
Applying log(x) transformation:



Applying z-norm transformation:

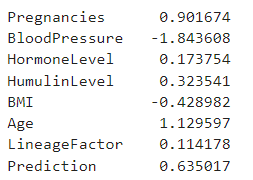


Applying quadratic transformation:



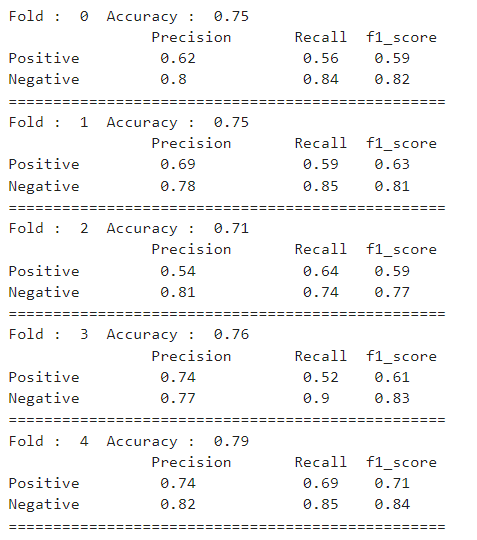
Here also, we can see that the transformation that gives the distribution closest to the gaussian distribution is the log(x) function. Hence, for ‘LineageFactor’ also, we choose the log(x) transformation.

The skewness of the features after transformation:

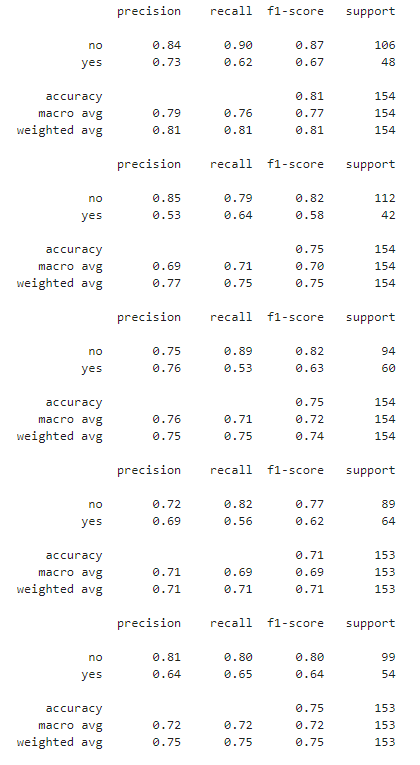


The results on the modified dataset are as follows:

**Our model:**

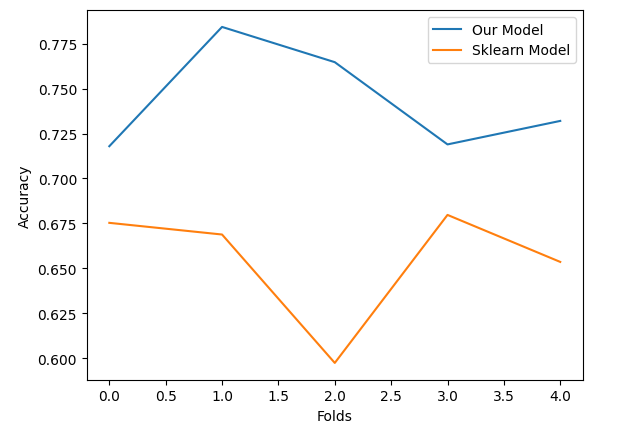


**Sklearn model**:

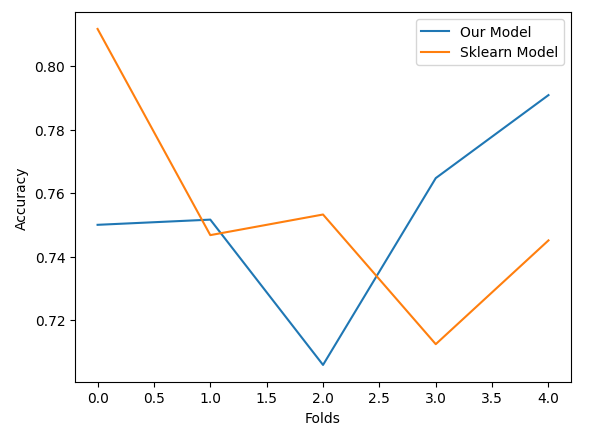


The comparative performance of both the scratch algorithm and sklearn is given as follows:

I. Before applying feature transformation:



Ii. After applying feature transformation:



**Average scores:**

1. **Before applying transformation:**

**Our Model:**

Average accuracy: 0.7435897435897436

Average precision (class 0): 0.7499101356262159

Average precision (class 1): 0.7239349039672024

Average recall (class 0): 0.9116639845566468

Average recall (class 1): 0.42519841269841263

Average F1-score (class 0): 0.8222103177651953

Average F1-score (class 1): 0.529143597828741

**Sklearn Model:**

Average accuracy: 0.6549783549783549

Average precision (class 0): 0.6569247265633702

Average precision (class 1): 0.61

Average recall (class 0): 0.9846671169293998

Average recall (class 1): 0.03921966398381493

Average F1-score (class 0): 0.7872657912098122

Average F1-score (class 1): 0.0731650805335016

1. **After applying transformation:**

**Our Model:**

Average accuracy: 0.7526143790849673

Average precision (class 0): 0.7966719365424269

Average precision (class 1): 0.6659355899164107

Average recall (class 0): 0.8348938043717744

Average recall (class 1): 0.5994918810436053

Average F1-score (class 0): 0.814111707440382

Average F1-score (class 1): 0.6268452797176581

**Sklearn Model:**

Average accuracy: 0.7538409303115186

Average precision (class 0): 0.7947943245695839

Average precision (class 1): 0.6703390344710287

Average recall (class 0): 0.8387524478332284

Average recall (class 1): 0.6023677248677248

Average F1-score (class 0): 0.8144340099810968

Average F1-score (class 1): 0.62902898701761