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:
    a. 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.

    b. 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.

    c. The 'Prediction' column is then dropped from both dataframes as it is not needed for further processing.

    d. 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:
    a. 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.

b. A list y_pred is then initialized to store the predicted class labels for each instance.

c. 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.

d. 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.

e. Finally, the method returns the predicted class labels stored in y_pred and the true class labels stored in y_test.

5. Score calculations:
   a. We wrote our own function imitating the same behavior as library function to generate performance score.
   b. Given the actual target values (y_test) and predicted target values (y_pred), it calculates the following evaluation metrics:

      i. True positive (TP)
      ii. False positive (FP)
      iii. True negative (TN)
      iv. False negative (FN)
      v. Accuracy: (TP + TN) / (TP + TN + FP + FN)
      vi. Precision (positive class): (TP) / (TP + FP)
      vii. Recall (positive class): (TP) / (TP + FN)
      viii. F1-Score (positive class): 2 / ((1/Precision) + (1/Recall))
      ix. Precision (negative class): TN / (TN + FN)
      x. Recall (negative class): TN / (FP + TN)
      xi. F1-Score (negative class): 2 / ((1/Precision) + (1/Recall))
   c. It then stores these metrics in a dictionary and returns the report.

Results:

The results on the validation sets are as follows:

Our model:

```
Fold :  0  Accuracy :  0.72
               Precision       Recall  f1_score
Positive         0.73            0.3     0.42
Negative         0.72            0.94    0.81
================================================
Fold :  1  Accuracy :  0.78
               Precision       Recall  f1_score
Positive         0.84            0.48    0.61
Negative         0.77            0.95    0.85
================================================
Fold :  2  Accuracy :  0.76
               Precision       Recall  f1_score
Positive         0.79            0.48    0.6
Negative         0.76            0.93    0.83
================================================
Fold :  3  Accuracy :  0.72
               Precision       Recall  f1_score
Positive         0.6             0.31    0.41
Negative         0.74            0.9     0.82
================================================
Fold :  4  Accuracy :  0.73
               Precision       Recall  f1_score
Positive         0.66            0.55    0.6
Negative         0.76            0.84    0.8
================================================
```

*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:

	precision	recall	f1-score	support
no	0.69	0.97	0.80	106
yes	0.25	0.02	0.04	48
accuracy			0.68	154
macro avg	0.47	0.50	0.42	154
weighted avg	0.55	0.68	0.57	154

	precision	recall	f1-score	support
no	0.66	1.00	0.80	101
yes	1.00	0.04	0.07	53
accuracy			0.67	154
macro avg	0.83	0.52	0.44	154
weighted avg	0.78	0.67	0.55	154

	precision	recall	f1-score	support
no	0.59	1.00	0.74	88
yes	1.00	0.06	0.11	66
accuracy			0.60	154
macro avg	0.79	0.53	0.43	154
weighted avg	0.76	0.60	0.47	154

	precision	recall	f1-score	support
no	0.68	0.99	0.80	101
yes	0.80	0.08	0.14	52
accuracy			0.68	153
macro avg	0.74	0.53	0.47	153
weighted avg	0.72	0.68	0.58	153

	precision	recall	f1-score	support
no	0.67	0.96	0.79	104
yes	0.00	0.00	0.00	49
accuracy			0.65	153
macro avg	0.34	0.48	0.40	153
weighted avg	0.46	0.65	0.54	153

Skewness:

The skewness of the original features is given by:

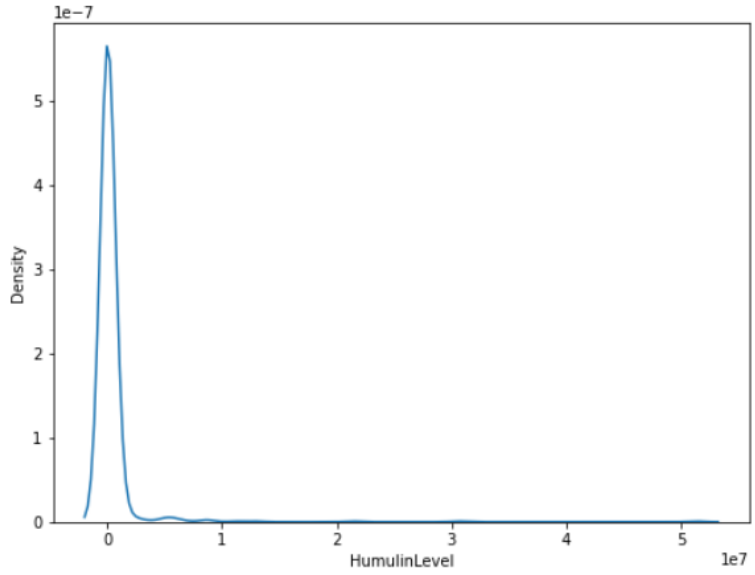
Pregnancies	0.901674
BloodPressure	-1.843608
HormoneLevel	0.173754
HumulinLevel	14.617070
BMI	-0.428982
Age	1.129597
LineageFactor	8.428404
Prediction	0.635017

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

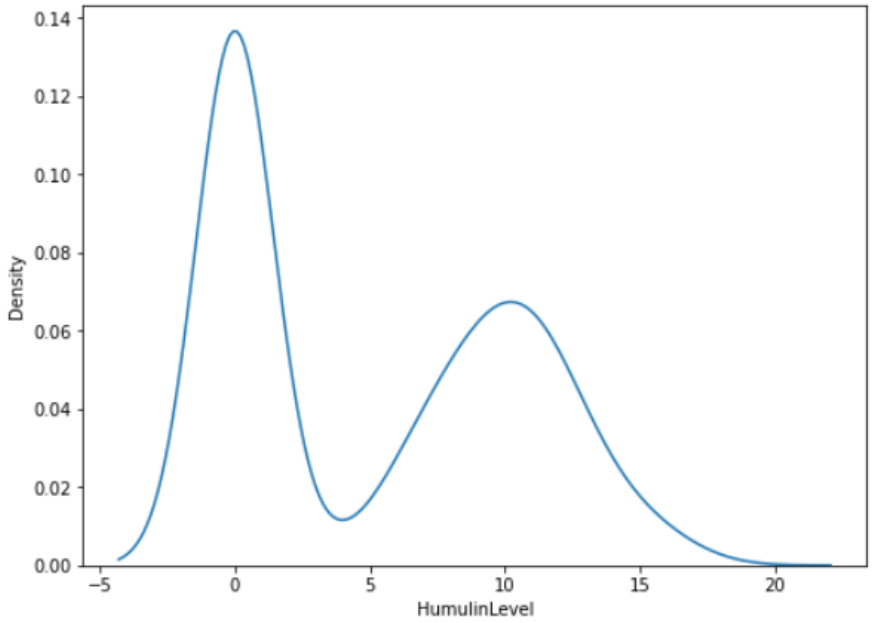
Feature Transformation

1. HumulinLevel:

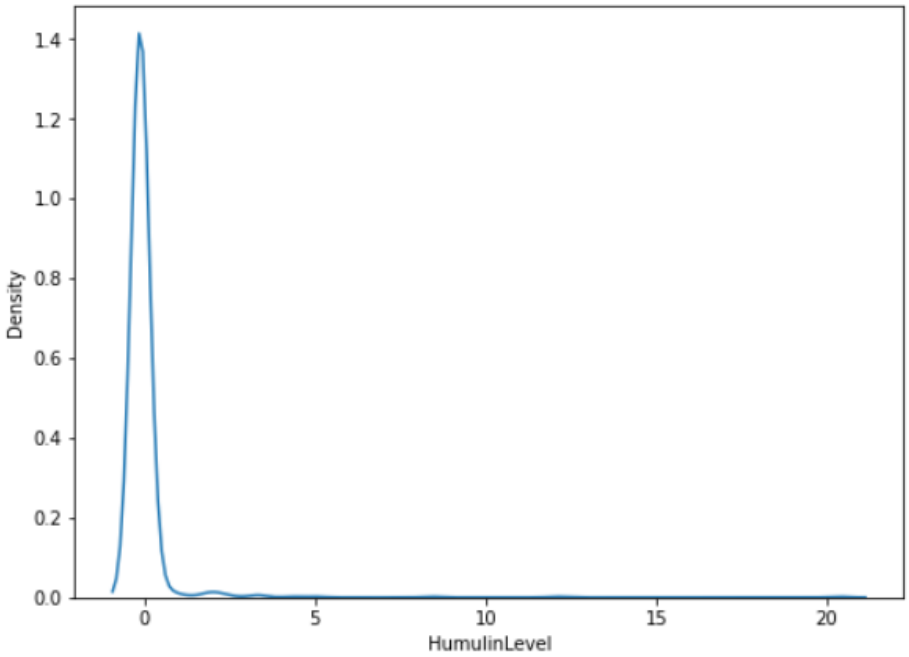
The kde-plot of ‘HumulinLevel’ is as shown:



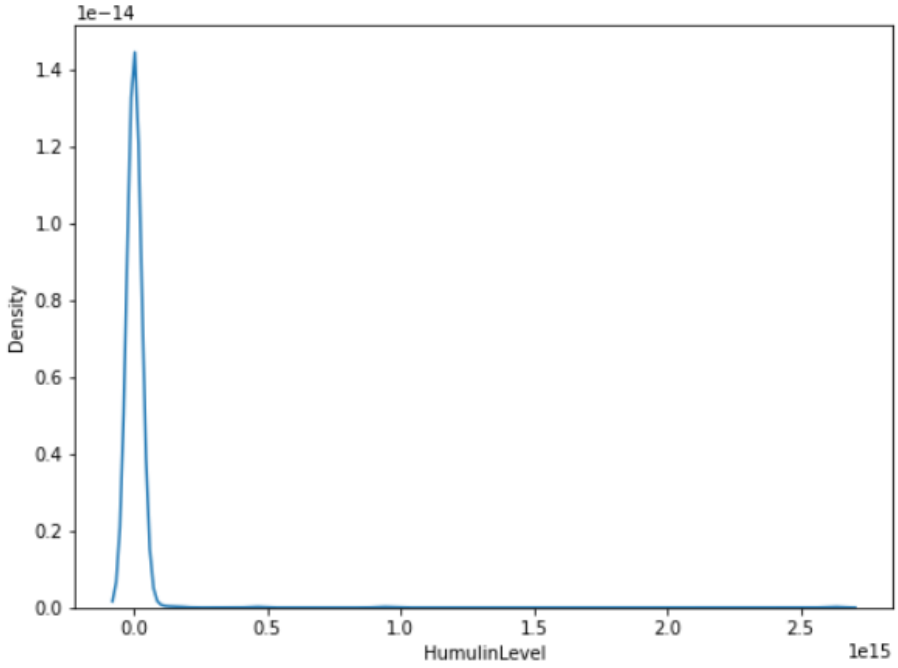
Applying log(1+x) transformation:



Applying z-norm transformation:



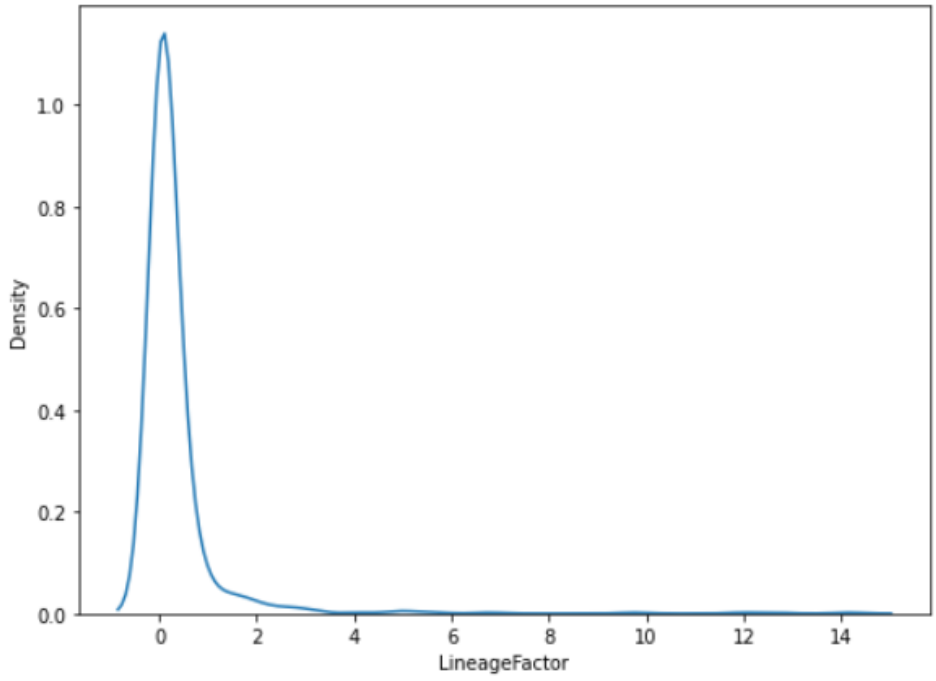
Applying quadratic transformation:



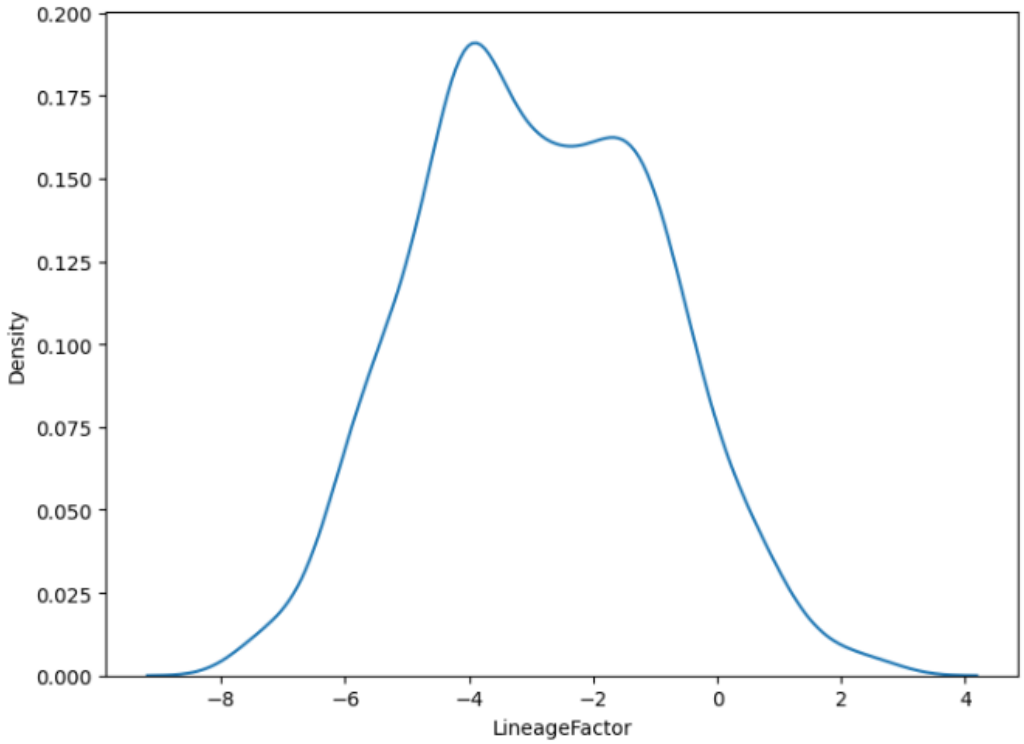
Among this, we can see that the log(1+x) transformation gives a plot closest to a bell-shaped curve. Hence log(1+x) was chosen as the function for transforming 'HumulinLevel'

2. LineageFactor:

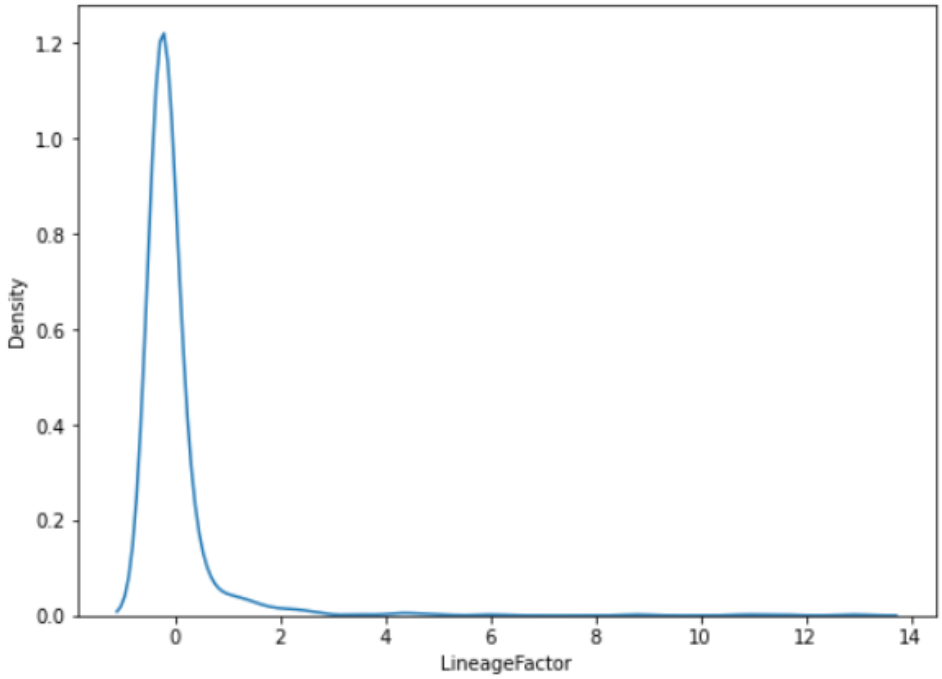
The kde-plot of 'LineageFactor' is as shown:



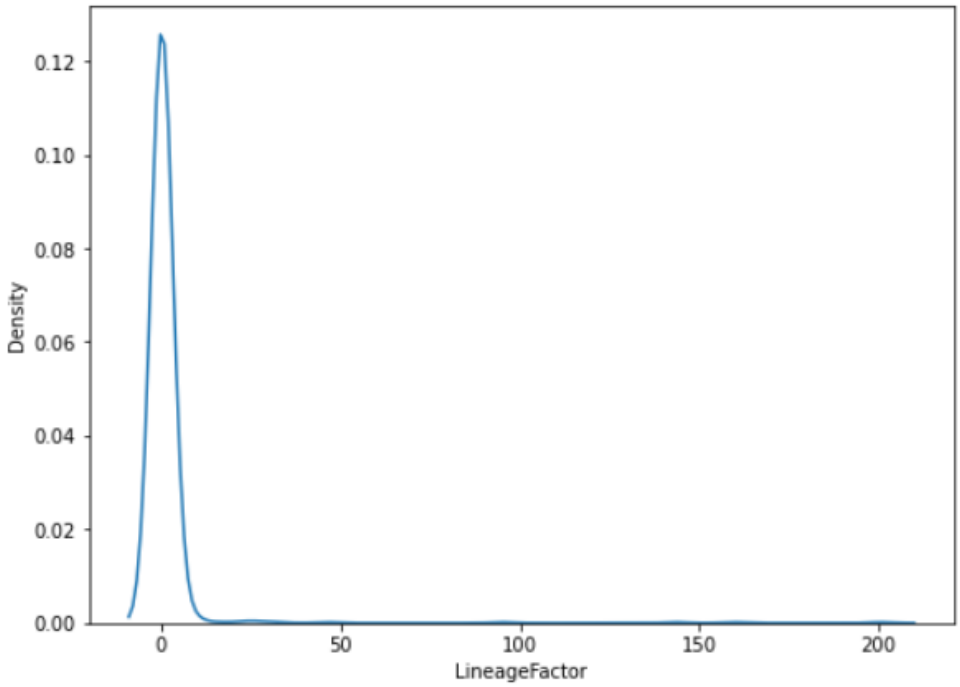
Applying log(x) transformation:



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:

Pregnancies	0.901674
BloodPressure	-1.843608
HormoneLevel	0.173754
HumulinLevel	0.323541
BMI	-0.428982
Age	1.129597
LineageFactor	0.114178
Prediction	0.635017

The results on the modified dataset are as follows:

Our model:

```
Fold :  0  Accuracy :  0.75
               Precision       Recall  f1_score
Positive        0.62            0.56    0.59
Negative        0.8             0.84    0.82
================================================
Fold :  1  Accuracy :  0.75
               Precision       Recall  f1_score
Positive        0.69            0.59    0.63
Negative        0.78            0.85    0.81
================================================
Fold :  2  Accuracy :  0.71
               Precision       Recall  f1_score
Positive        0.54            0.64    0.59
Negative        0.81            0.74    0.77
================================================
Fold :  3  Accuracy :  0.76
               Precision       Recall  f1_score
Positive        0.74            0.52    0.61
Negative        0.77            0.9     0.83
================================================
Fold :  4  Accuracy :  0.79
               Precision       Recall  f1_score
Positive        0.74            0.69    0.71
Negative        0.82            0.85    0.84
================================================
```

Sklearn model:

	precision	recall	f1-score	support
no	0.84	0.90	0.87	106
yes	0.73	0.62	0.67	48
accuracy			0.81	154
macro avg	0.79	0.76	0.77	154
weighted avg	0.81	0.81	0.81	154

	precision	recall	f1-score	support
no	0.85	0.79	0.82	112
yes	0.53	0.64	0.58	42
accuracy			0.75	154
macro avg	0.69	0.71	0.70	154
weighted avg	0.77	0.75	0.75	154

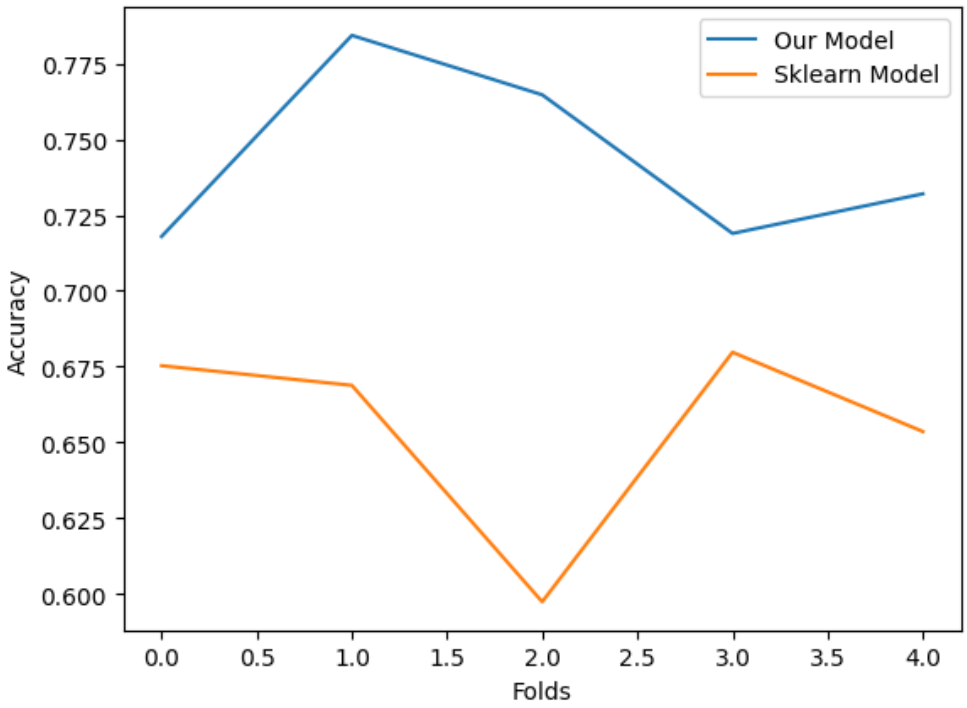
	precision	recall	f1-score	support
no	0.75	0.89	0.82	94
yes	0.76	0.53	0.63	60
accuracy			0.75	154
macro avg	0.76	0.71	0.72	154
weighted avg	0.75	0.75	0.74	154

	precision	recall	f1-score	support
no	0.72	0.82	0.77	89
yes	0.69	0.56	0.62	64
accuracy			0.71	153
macro avg	0.71	0.69	0.69	153
weighted avg	0.71	0.71	0.71	153

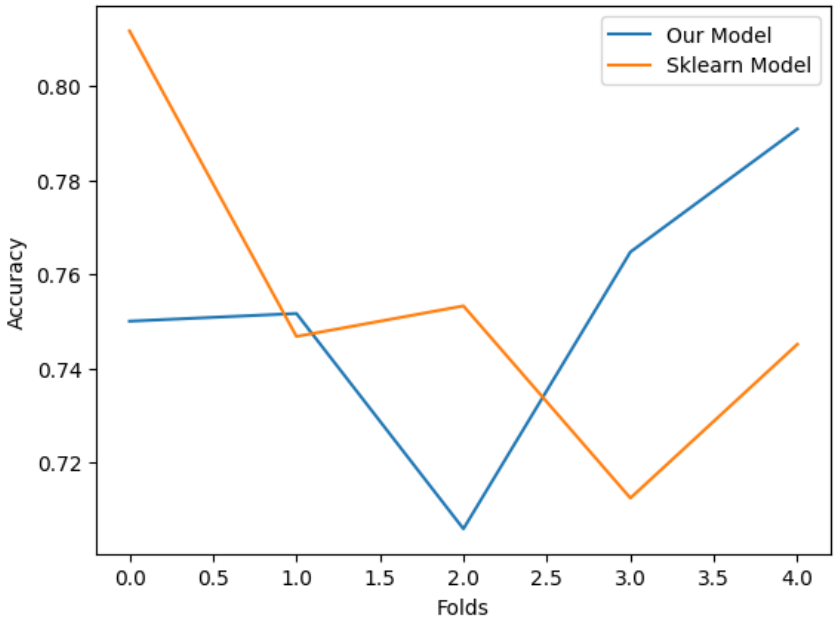
	precision	recall	f1-score	support
no	0.81	0.80	0.80	99
yes	0.64	0.65	0.64	54
accuracy			0.75	153
macro avg	0.72	0.72	0.72	153
weighted avg	0.75	0.75	0.75	153

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

2. **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