Evaluation of K Nearest Neighbors algorithm with Sklearn

Pritam Mishra
ID 35390350

***Abstract*— Data mining although still in its infancy, organisations in a wide range of industries - including but not limited to retail, finance, heath care, manufacturing, transportation, aerospace - are already applying data mining applications and machine learning algorithms to take advantage of historical data. K nearest Neighbors is one of the most popular algorithms which has a high efficacy & intuitive by nature in terms of implementation. In this project, K Nearest Neighbors algorithm has been implemented from scratch in python & analysed in comparison with in built KNN of Sklearn library. The performance metrics of both the algorithms have been compared for unique values of parameter k, which in turn is analysed further in R with line plots & hypothesis testing. The idea is to analyse if the algorithm from scratch performs better in comparison with Sklearn's KNN for a given dataset on several test cases.**

I. INTRODUCTION

The KNN algorithm is a robust classifier which is often used as a benchmark for more complex classifiers such as Artificial Neural Network or Support vector machine. It is a non-parametric algorithm which means there are no assumptions to be met compared to algorithms like linear regression which has lots of assumptions. It has the flexibility for the user to choose from any of the distance based approach for the modelling,

- Euclidean Distance
- Hamming Distance
- Manhattan Distance
- Minkowski Distance

In this project, Euclidean distance has been applied for modelling with KNN algorithm. The data-set is scaled & encoded before applying KNN algorithm. Also cross-validation technique has been implemented in this project to increase the robustness of the algorithm, as all the performance metrics has been determined as the average of those metrics for all k folds where k is the number of folds in cross-validation. The computational time although a little higher compared to the Sklearn counterpart, the KNN algorithm from scratch produces a much robust performance & very consistent results when compared with Sklearn's algorithm.

II. METHODS

The following methodology has been implemented in this project to compare the performance of KNN from scratch with Sklearn's KNN algorithm & further analysed each performance metric with a hypothesis.

A. Handling Categorical Data

Data-sets capturing aspects of real world might contain categorical values along with numerical data. Algorithms like decision Trees can handle categorical data but most of the algorithms expect numerical values to achieve state-of-the-art results. Therefore Encoding technique is implemented to convert the categorical data into numerical values.

1) Label Encoder: This approach is very simple and it involves converting each value in a column to a number. In this project, Label Encoder has been applied to convert categorical data present in the mushroom data-set to convert into numerical values & compared KNN algorithms later based on the encoded data.

B. Scaling

Usually, most data-sets contain features highly varying in magnitudes, units & range. However, since most of the machine learning algorithms like KNN determines euclidean distance between data points, this creates erroneous computation of distance which leads to inaccurate results. The results can vary greatly between different units for example, 1000 metres & 1 km. If not scaled 1000 metres will weight a lot more than 1 km for this example.

1) Standardisation: The result of standardization (Z-score normalization) is that the features will be re-scaled so that they'll have the properties of a standard normal distribution with

$$\mu = 0 \;\; \sigma = 1 \tag{1}$$

where μ is the mean (average) and σ is the standard deviation from the mean; standard scores (also called z scores) of the samples are calculated as follows:

$$z = \frac{x - \mu}{\sigma} \tag{2}$$

Standardizing the features so that they are centered around 0 with a standard deviation of 1 is not only important for measurements that have different units, but also a general requirement for many machine learning algorithms like KNN, otherwise there could be erroneous results.

C. Cross-validation

K-Fold Cross-validation is a popular and easy to understand approach in cross-validation, it generally results in a less biased model compare to other methods. Because it ensures that every observation from the original data-set has the chance of appearing in training and test set. In this project k fold cross-validation has been implemented manually in order to produce a robust performance metric for both KNN classifiers.This method follows the below steps,

- Split the entire data randomly into k folds (value of k shouldn't be too small or too high, ideally 5 to 10 is

chosen depending on the data size). The higher value of K leads to less biased model (but large variance might lead to overfit),
- Then fit the model using the K - 1 (K minus 1) folds and validate the model using the remaining Kth fold.
- This process is repeated until every fold is served as a test set. Finally the average performance metric(Accuracy,precision,recall etc.) of all the validation sets are recorded for evaluation.

D. Performance Metrics

1) Accuracy: It is usually meant by the accuracy of the model provided there is no class imbalance in the data-set. In case there is class imbalance in the data-set, the data-set can be over-sampled or under-sampled to handle the imbalance between the classes. The alternative way to analyze the accuracy of the model is to obtain the other advanced performance measures which will be discussed soon. The accuracy here can be defined as ratio of number of correct predictions to the total number of input samples.

$$Accuracy = \frac{CorrectPrediction}{Total\ number\ of\ predictions} \tag{3}$$

2) Confusion Matrix: A confusion matrix is a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known. There are four sections in a confusion matrix which can be defined by the four terms as follows,

- **True Positives:** The cases in which the predicted output is YES and the actual output was also YES
- **True Negatives:** The cases in which the predicted output is NO and the actual output was also NO.
- **False Positives:** The cases in which the predicted output is YES and the actual output was NO.
- **False Negatives:** The cases in which the predicted output is NO and the actual output was YES.

3) Area Under Curve: Area Under Curve(AUC) is one of the most widely used metrics for evaluation. It is used for binary classification problem. AUC of a classifier is equal to the probability that the classifier will rank a randomly chosen positive example higher than a randomly chosen negative example.

- **True Positive Rate (Sensitivity):** True Positive Rate is defined as TP/(FN+TP). True Positive Rate corresponds to the proportion of positive data points that are correctly considered as positive, with respect to all positive data points.

$$True\ Positive\ Rate = \frac{True\ Positive}{False\ Negative + True\ Positive} \tag{4}$$

- **False Positive Rate (Specificity):** False Positive Rate is defined as FP/(FP+TN). False Positive Rate corresponds to the proportion of negative data points that are mistakenly considered as positive, with respect to all negative data points.

$$False\ Positive\ Rate = \frac{False\ Positive}{False\ Positive + True\ Negative} \tag{5}$$

False Positive Rate and True Positive Rate both have values in the range [0, 1]. FPR and TPR both are computed at threshold values such as (0.00, 0.02, 0.04, …., 1.00) and a graph is drawn. AUC is the area under the curve of plot False Positive Rate vs True Positive Rate at different points in [0, 1].

4) F1 Score: F1 Score is defined as the Harmonic Mean between precision and recall.

5) Precision :: It is the number of correct positive results divided by the number of positive results predicted by the classifier.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{6}$$

6) Recall :: It is the number of correct positive results divided by the number of all samples that should have been positive.

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{7}$$

Now the F1 score can be defined as,

$$F1 = 2 * \frac{1}{\frac{1}{Precision} + \frac{1}{Recall}} \tag{8}$$

F1 Score tries to find the balance between precision and recall. In short, the greater the F1 Score, the better is the performance of the given model.

7) Processing Time: The processing time of each Algorithm has been recorded for different values of parameter k. Here, the processing time of each algorithm is also being considered as one of the performance metric & further exported in a csv file.

E. Exporting CSV file

In this project, the average of all the performance metrics have been recorded for each fold in a k-fold cross-validation. Further, these metrics have been exported to a csv file so that these metrics can be plotted in R using graphical tools like ggplot2 & further analysed & compared with hypothesis testing methods like T-test.

F. Analysis with R

The exported csv file is analysed in R language to have a detailed comparison between the performance metrics of these two KNN algorithms with varying parameter k. First, for each performance metric comparison between the KNN from scratch & it's Sklearn counterpart has been demonstrated with the help of line plots in ggplot2. Further, hypothesis testing has been done for each metric of the performance to test whether the null or alternate hypothesis is being true.

1) T-Test: One of the most common tests in statistics is the t-test, used to determine whether the means of two groups are equal to each other. The assumption for the test is that both groups are sampled from normal distributions. The null hypothesis is that the two means are equal, and the alternative is that they are not. In order to perform this test, first F-test needs to be performed between the performance metric data

of two KNN algorithm to find out if the variance ratio of these two distribution is same or not.

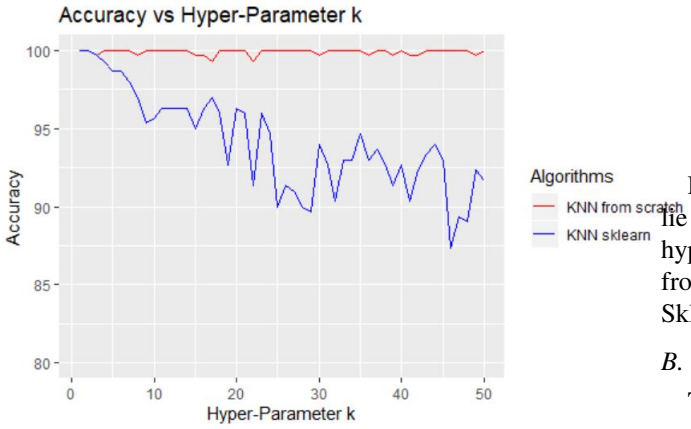
For example, In order to compare the hypothesis regarding performance metric Accuracy, first the variance ratio was determined between the accuracy distribution of data for two KNN algorithms. Once it is determined the variance ration is not equal to one, T-Test was performed to evaluate the hypothesis for accuracy. The alternate hypothesis H1 for this case was - The mean accuracy of KNN from Scratch is greater than KNN from sklearn. The Null hypothesis H0 for this test was- The mean accuracy of KNN from Sklearn is greater than KNN from scratch. After performing the T-Test, it was proved that our alternate hypothesis was true, as the value of T statistic lie within the five percent critical region.

III. Results

In this section, the detailed comparison & analysis between the KNN from scratch & KNN from Sklearn is discussed, as the result for each performance metric is shown below for the experiment performed on a specific data-set.

A. Accuracy

The Accuracy comparison plot of both algorithms for increasing value of hyper-parameter k in KNN algorithm is shown as follows,



From the above plot, it is clear that the accuracy of KNN from scratch algorithm shows consistent results as the there is not much variation in terms of accuracy with increase in value of k. When compared with the accuracy of KNN from SKlearn, it performs inconsistently, while the accuracy is very high at the beginning it goes to as low as below 90 percent as the value of k increases.

Now, in order to be sure that KNN from scratch is performing better than the Sklearn counterpart, the average of these two accuracy distribution can be compared with a paired T-Test. But before performing T-Test, the variance ratio of these two distribution needs to be determined with an F-Test.

```
    F test to compare two variances

data:  data_scratch$Accuracy and data_sklearn$Accuracy
F = 0.0032412, num df = 49, denom df = 49, p-value < 2.2e-16
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.001839300 0.005711592
sample estimates:
ratio of variances
       0.003241193
```

Now, The mean accuracy of both the algorithms can be compared with a T-Test. The alternate hypothesis for the test H1: The mean accuracy of KNN from scratch is greater than KNN from Sklearn & Null hypothesis for the test H0: The mean accuracy of KNN from scratch is lesser than KNN from sklearn.

```
a <- var(data_scratch$Accuracy)
b <- var(data_sklearn$Accuracy)

nu <- (((a/50) + (b/50)) ^ 2) / (((a/50)^2)/49) + (((b/50)^2)/49)
```

```
t.test(data_scratch$Accuracy, data_sklearn$Accuracy, alternative = "greater", pa
```

```
    Paired t-test

data:  data_scratch$Accuracy and data_sklearn$Accuracy
t = 13.07, df = 49, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 5.067621      Inf
sample estimates:
mean of the differences
               5.813333
```

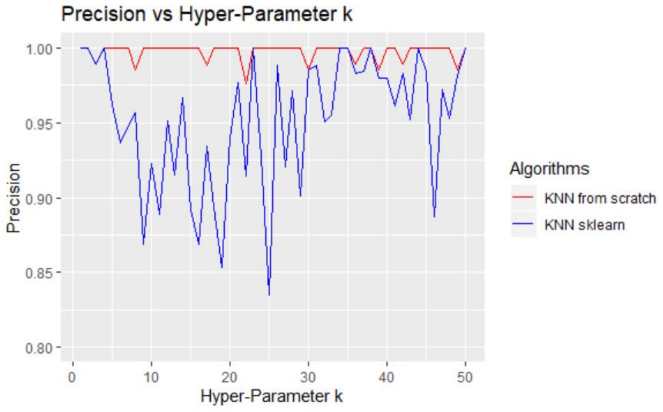
```
#c(qt(0.025, nu),qt(0.975, nu))
c(qt(0.05, nu),qt(0.95, nu))
```

```
[1] -1.644854  1.644854
```

From the figure, it is clear that the value of T statistics lie within the critical region, hence we can reject the NULL hypothesis H0 & conclude that the mean accuracy of KNN from scratch is greater than mean accuracy of KNN from Sklearn.

B. Precision

The precision comparison plot for both the algorithms are shown below,



The results of F-Test & T-Test with NULL & alternate Hypothesis is shown as follows,

```
H0:KNN from scratch has lower mean precision compared to sklearn
counterpart
H1:KNN from scratch has greater mean precision compared to sklearn
counterpart
```

```{r}
t.test(data_scratch$Precision, data_sklearn$Precision, alternative =
"greater", paired = T)
```

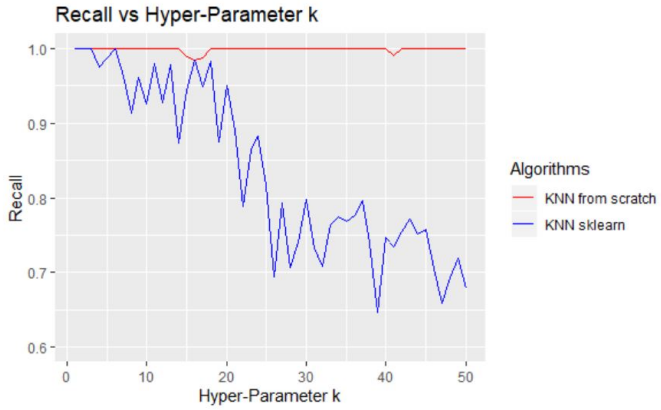
```
        Paired t-test

data:  data_scratch$Precision and data_sklearn$Precision
t = 7.4831, df = 49, p-value = 5.942e-10
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.06995177        Inf
sample estimates:
mean of the differences
             0.09014926
```

```{r}
c(qt(0.05, nu),qt(0.95, nu))
```

```
[1] -1.644857  1.644857
```

C. Recall

The Recall comparison plot of both algorithms for increasing value of hyper-parameter k in KNN algorithm is shown as follows,



The results of F-Test & T-Test with NULL & alternate Hypothesis is shown as follows,

```
H0:KNN from scratch has lower mean recall compared to sklearn counterpart
H1:KNN from scratch has greater mean recall compared to sklearn
counterpart
```

```{r}
t.test(data_scratch$Recall, data_sklearn$Recall, alternative = "greater",
paired = T)
```

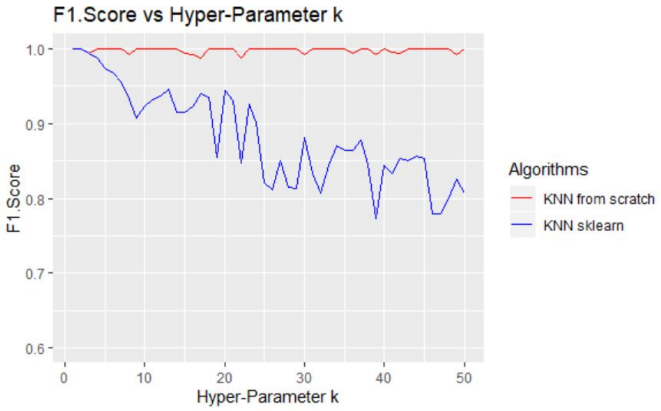
```
        Paired t-test

data:  data_scratch$Recall and data_sklearn$Recall
t = -0.86607, df = 49, p-value = 0.8047
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -0.004110139         Inf
sample estimates:
mean of the differences
                -0.0014
```

```{r}
c(qt(0.05, nu),qt(0.95, nu))
```

```
[1] -1.644857  1.644857
```

D. F1 Score

The F1 Score comparison plot of both algorithms for increasing value of hyper-parameter k in KNN algorithm is shown as follows,



The results of F-Test & T-Test with NULL & alternate Hypothesis is shown as follows,

```
H0:KNN from scratch has lower mean f1 score compared to sklearn
counterpart
H1:KNN from scratch has greater mean f1 score compared to sklearn
counterpart
```

```{r}
t.test(data_scratch$F1.Score, data_sklearn$F1.Score, alternative =
"greater", paired = T)
```

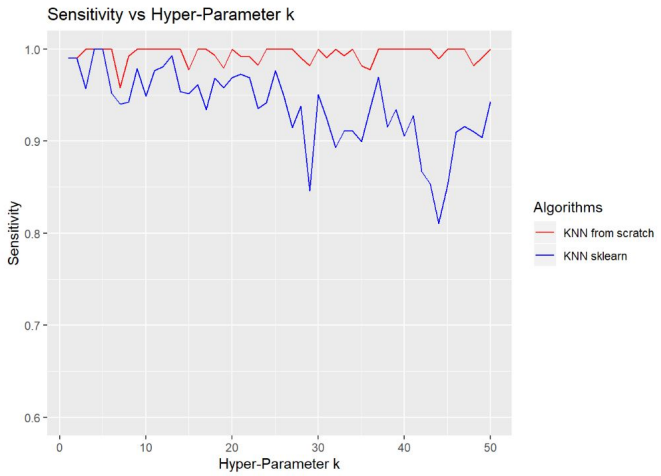
```
        Paired t-test

data:  data_scratch$F1.Score and data_sklearn$F1.Score
t = 6.9077, df = 49, p-value = 4.609e-09
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.037976      Inf
sample estimates:
mean of the differences
             0.05014698
```

```{r}
c(qt(0.05, nu),qt(0.95, nu))
```

```
[1] -1.644943  1.644943
```

E. Sensitivity

The Sensitivity comparison plot of both algorithms for increasing value of hyper-parameter k in KNN algorithm is shown as follows,



The results of F-Test & T-Test with NULL & alternate Hypothesis is shown as follows,

```
H0:KNN from scratch has lower mean f1 score compared to sklearn
counterpart
H1:KNN from scratch has greater mean f1 score compared to sklearn
counterpart
```

```{r}
t.test(data_scratch$Sensitivity, data_sklearn$Sensitivity, alternative =
"greater", paired = T)
```

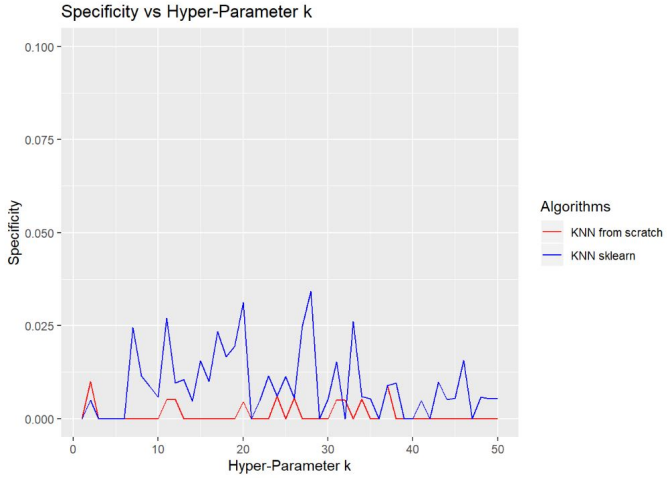
```
        Paired t-test

data:  data_scratch$Sensitivity and data_sklearn$Sensitivity
t = -0.86607, df = 49, p-value = 0.8047
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -0.004110139          Inf
sample estimates:
mean of the differences
                -0.0014
```

```{r}
c(qt(0.05, nu),qt(0.95, nu))
```

```
[1] -1.671872  1.671872
```

F. Specificity

The Specificity comparison plot of both algorithms for increasing value of hyper-parameter k in KNN algorithm is shown as follows,



The results of F-Test & T-Test with NULL & alternate Hypothesis is shown as follows,

```
H0:KNN from scratch has greater mean specificity compared to sklearn
counterpart
H1:KNN from scratch has lower mean specificity compared to sklearn
counterpart
```

```{r}
t.test(data_scratch$F1.Score, data_sklearn$F1.Score, alternative =
"greater", paired = T)
```

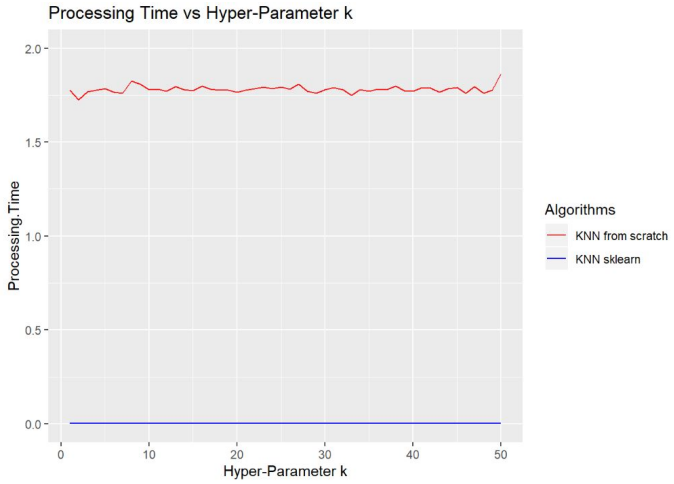
```
        Paired t-test

data:  data_scratch$F1.Score and data_sklearn$F1.Score
t = 6.9077, df = 49, p-value = 4.609e-09
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.037976      Inf
sample estimates:
mean of the differences
             0.05014698
```

```{r}
c(qt(0.05, nu),qt(0.95, nu))
```

```
[1] -1.671872  1.671872
```

G. Processing Time

The Processing time comparison plot of both algorithms for increasing value of hyper-parameter k in KNN algorithm is shown as follows,

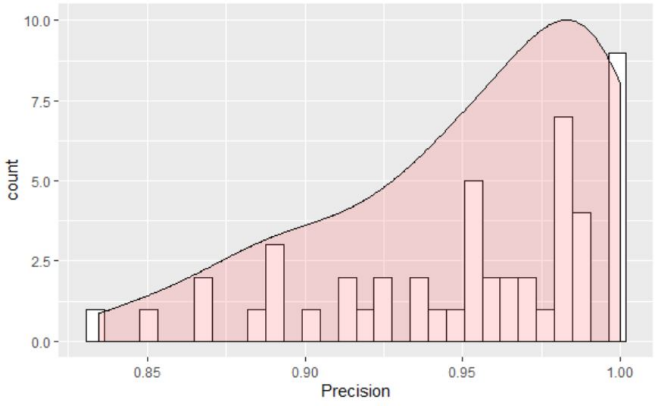


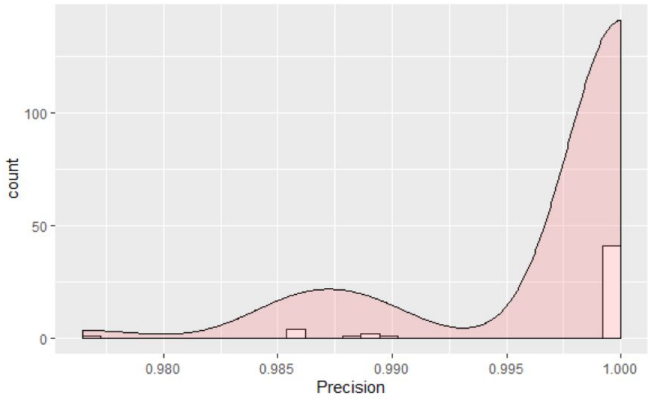
IV. Discussion

The K nearest Neighbors algorithm from scratch is showing a better performance overall in terms of all performance metrics except processing time. While Sklearn's KNN algorithm is showing very high accuracy & minimal processing time, it fails to provide the robustness of KNN algorithm as the number of neighbor increases. The performance takes a hit for sklearn's KNN algorithm when it is compared with the KNN scratch implementation at a value between 30-40. It is also impressive how KNN from scratch algorithm maintains excellent precision,recall,f1 score for all values of k, if this algorithm is optimized further the processing time can get closer compared to sklearn's classifier.

APPENDIX

Histogram for Precision of both algorithms can be shown below,





The exported csv file is shown below that has multiple performance metrics for each of the classifier.

	A	B	C	D	E	F	G	H	I	J
1		Identifier	Number of Neighbors	Precision	Recall	F1 Score	Sensitivity	Specificity	Processing Time	Accuracy
2	0	KNN FROM SCRATCH	1	1	0.99047619	0.995121951	0.99047619	0	1.77638426	99.66666667
3	1	KNN SKLEARN	1	1	0.99047619	0.995121951	0.99047619	0	0.00143618	99.66666667
4	2	KNN FROM SCRATCH	2	0.980952381	0.99	0.985365854	0.99	0.01	1.72473768	99
5	3	KNN SKLEARN	2	0.99	0.99	0.99	0.99	0.005	0.00135874	99.33333333
6	4	KNN FROM SCRATCH	3	1	1	1	1	0	1.76833126	100
7	5	KNN SKLEARN	3	1	0.956814815	0.977734845	0.956814815	0	0.00118084	98.66666667
8	6	KNN FROM SCRATCH	4	1	1	1	1	0	1.77813752	100
9	7	KNN SKLEARN	4	1	1	1	1	0	0.0012904	100
10	8	KNN FROM SCRATCH	5	1	1	1	1	0	1.78531398	100
11	9	KNN SKLEARN	5	1	1	1	1	0	0.00129888	100
12	10	KNN FROM SCRATCH	6	1	1	1	1	0	1.76614626	100
13	11	KNN SKLEARN	6	1	0.952206603	0.974765535	0.952206603	0	0.00130214	98.33333333
14	12	KNN FROM SCRATCH	7	1	0.957894737	0.978470588	0.957894737	0	1.76093468	98.66666667
15	13	KNN SKLEARN	7	0.949144385	0.94011544	0.942727097	0.94011544	0.024426921	0.0014018	96.33333333
16	14	KNN FROM SCRATCH	8	1	0.992307692	0.996078431	0.992307692	0	1.824834	99.66666667
17	15	KNN SKLEARN	8	0.983666667	0.942502787	0.962445039	0.942502787	0.011437908	0.00127742	97
18	16	KNN FROM SCRATCH	9	1	1	1	1	0	1.80893994	100
19	17	KNN SKLEARN	9	0.973333333	0.978666667	0.975688482	0.978666667	0.008792271	0.00135506	98.66666667
20	18	KNN FROM SCRATCH	10	1	1	1	1	0	1.7800245	100
21	19	KNN SKLEARN	10	0.992	0.948888889	0.96889441	0.948888889	0.005714286	0.00143284	97.66666667
22	20	KNN FROM SCRATCH	11	0.990909091	1	0.995348837	1	0.005128205	1.78156766	99.66666667
23	21	KNN SKLEARN	11	0.956912972	0.976436782	0.966119418	0.976436782	0.026984183	0.00136022	97.33333333
24	22	KNN FROM SCRATCH	12	0.991304348	1	0.995555556	1	0.005263158	1.7702958	99.66666667
25	23	KNN SKLEARN	12	0.978362573	0.980555556	0.979228164	0.980555556	0.00952381	0.00156904	98.66666667
26	24	KNN FROM SCRATCH	13	1	1	1	1	0	1.79499632	100
27	25	KNN SKLEARN	13	0.98247619	0.992592593	0.987266734	0.992592593	0.010555556	0.00145086	99
28	26	KNN FROM SCRATCH	14	1	1	1	1	0	1.78078426	100

REFERENCES

[1] Andreas C. Müller & Sarah Guido, ”Introduction to machine learning with Python”

[2] Rohith Gandhi, K Nearest Neighbours — "Introduction to Machine Learning Algorithms", ”https://towardsdatascience.com/k-nearest-neighbours-introduction-to-machine-learning-algorithms-18e7ce3d802a”

[3] Jason Brownlee, "Develop k-Nearest Neighbors in Python From Scratch", https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/