CHAPTER 5
IMPLEMENTATION AND TESTING

5.1 Implementation

5.1.1 Split Data

In this stage, it explains how to program and use the Python programming language. At this stage it also explains how the data classification begins with sharing data for training data and testing data. In data sharing, it is arranged based on the number of split data. The number of split data affects the amount of training and testing data.

The following is the code for split data.

```
1.with open('i_data_sample.csv','rb') as f:
2.    reader= unicodecsv.reader(f)
3.    i_data=list(reader)
4.def shuffle(i_data):
5.    random.shuffle(i_data)
6.    train_data = i_data[;int(0,7*30)]
7.    test_data = i_data[;int(0,7*30)]
8.    return train_data, test_data
```

Lines 1-3 contain commands for loading a dataset from a csv file. Lines 4-8 are functions for split data. Line 5 functions to randomize and shuffle the data. Line 6 functions train data by 70% and testing 30%. Line 8 to return train data and test data.

5.1.2 Euclidean Distance

The following is the program code for Euclidean Distance

```
9. def get_distance(x, xi):
10.    for i in range(len(x-1):
11.    d+=pow((float(x[i])-float(xi[i])),2)
12.    d= math.sqrt(d)
13.    return d
```

In lines 9 to 13, it functions to find the euclidean distance value. Euclidean distance is based on the distance between 2 objects. Line 10 serves for the loop of the variable i in the long range of values x-1. Line 11 functions to calculate the euclidean distance from the value x [i] minus xi [i] then raised to the power. Line 13 returns the root.

5.1.3 Distance

```
14.eu_Distance=[]
15.for j in train data:
16.    eu_dist = get_distance(i, j)
17.    eu_Distance.append(j[5], eu_dist))
18.    eu_Distance.sort(key=operator.itemgetter(1))
19.    knn=eu_Distance[:k_value}
```

Line 14 creates an empty list for the range of values. Line 15 serves to loop the variable j in the training data. Line 16 takes the value from the euclidean distance. Line 17 serves to take the closest distance from the euclidean distance value

5.1.4 Sorted Vote

```
20.  def get_voting(neighbours):
21.    # index 1 is the class
22.    classes = [neighbour[1] for neighbour in neighbours]
23.    count = Counter(classes)
24.    return count.most_common()[0][0]
```

Lines 20-25 have commands for sorting by class sound. On line 23, will count the amount of data in the class. On line 24 the aim is to return the count value. On line 22, you get a vote from the nearest neighbor based on index 1.

5.1.5 Predictions

```
26.  for k in knn:
27.         if k[0] == 'g':
28.                good +=1
29.         else
30.         bad +=1
31.   if good > bad:
32.         i.append('g')
33.   elif good < bad:
34.         i.append('b')
35.   else:
36.         i.append('NaN')
```

Lines 26-36 are steps or functions to get predictions. Line 26 is for loop k in knn. Lines 27-36 are the prediction function, i.e. if k [0] equals 'g' then good is added to 1 and if it

doesn't add bad 1. If good is greater than bad then it will print g and if good is less than bad then it will print b and if not both then empty.

5.1.6 Accuracy

```
37.    def compute_accuracy(tp, tn, fn, fp):
38.
39.        return (tp*100)/float(tp+fp)
40.        print('Accuracy:', compute_accuracy(tp,tn,fn,fp)
```

On lines 37-40 are functions to ensure accuracy. In line 39, the accuracy is obtained from the number of correct predictions with the actual correct (tp) multiplied by 100 which is the percentage divided by the number of tp plus fp, where fp is the number of correct predictions but the actual ones are late.

5.1.7 Gini Index

```
44.    def gini_index(groups, classes):
44.        n_point = float(sum([len(group) for group in
groups]))
46.        size = float(len(group))
47.        p = [row[-1] for row in group].count(class_val) /
size
48.        score += p * p
49.        gini += (1.0 - score) * (size / n_point)
```

On line 44 the function is to calculate the n_point value of adding the group length in the group group in groups loop.

5.2 Testing

5.2.1 Comparison Precision, Recall and F1 Score Without Add New Dataset

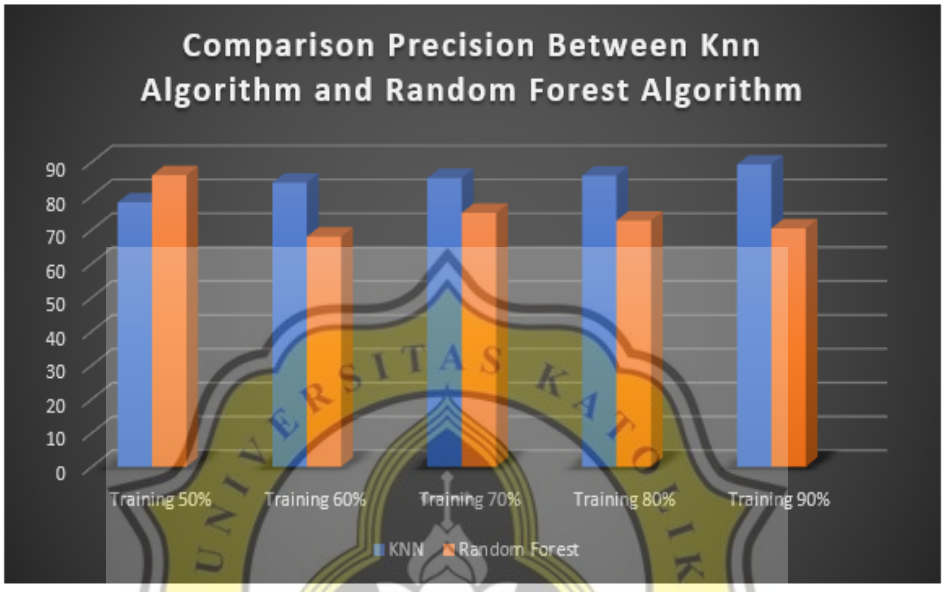

Illustration 16: Comparison Precision Between Knn Algorithms and Random Forest Algorithms with 372 data.

Based on Illustration 16, it shows that the precision results from both algorithms without adding a new dataset. If the precision results have a low value, there is a high probability that errors will occur in the tests performed. From the above results, the KNN algorithm gets higher results from the split data test of 60%, 70%, 80%, 90% compared to the random forest algorithm. While the Random Forest Algorithm only gets the greatest value on the split test by 50%. So it can be concluded that the KNN Algorithm has fewer errors in the tests carried out than the Random Forest algorithm because the KNN Algorithm gets a higher value than the Random Forest Algorithm.

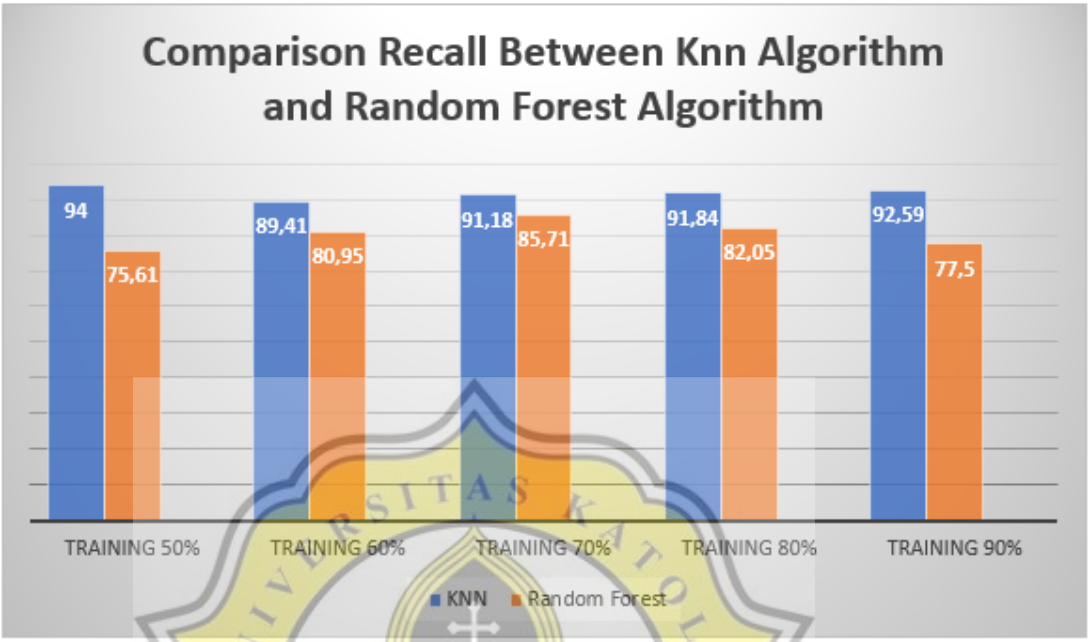

Illustration 17: Comparison Recall Between Knn Algorithms and Random Forest Algorithms with 372 data.

Illustration 17 shows that the recall results of both algorithms without adding a new dataset. Recall value is looking for how to get the acquisition or amount. The greater the recall value does not guarantee a better precision value. The smaller the False Negative (FN) value, the greater the recall value. From the results above, the KNN algorithm gets higher results from split data testing by 50%, 60%, 70%, 80%, 90% than the random forest algorithm. So it can be concluded that the KNN Algorithm has a better recall value than the Random Forest Algorithm because it has a higher recall value than the Random Forest algorithm.

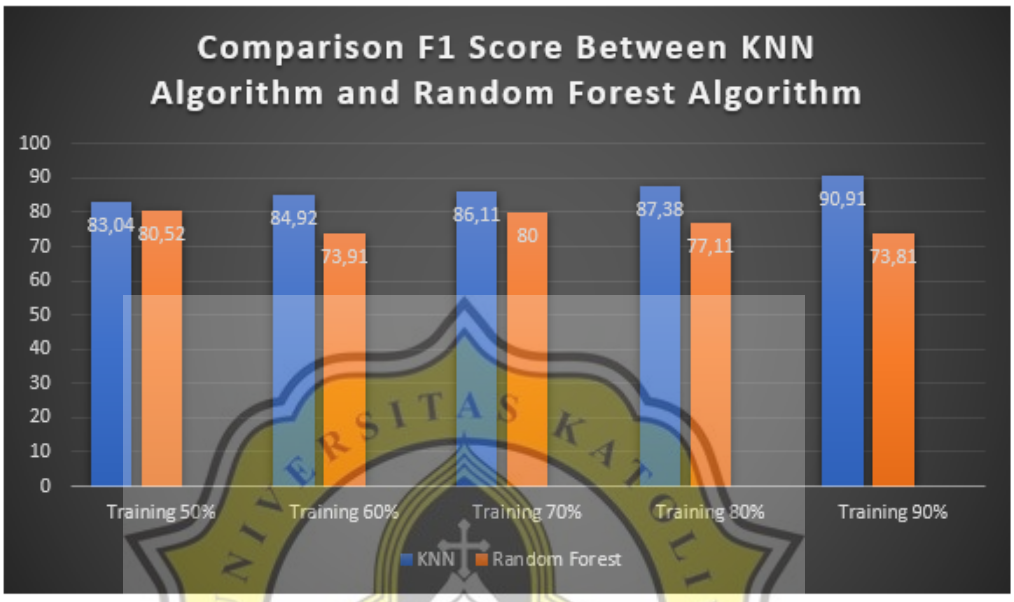

Illustration 18: Comparison F1 Score Between Knn Algorithms and Random Forest Algorithms with 372 data.

Based on illustration 18 it proves that the average precision and recall or f1 score of the KNN algorithm is better than the random forest algorithm. The f1 score value is needed to find a balance between precision and recall. From the results above, it can show that the KNN algorithm is better in the balance between precision and recall in testing training data by 50%, 60%, 70%, 80%, and 90%.

5.2.1 Comparison Precision, Recall and F1 Score With Add New Dataset

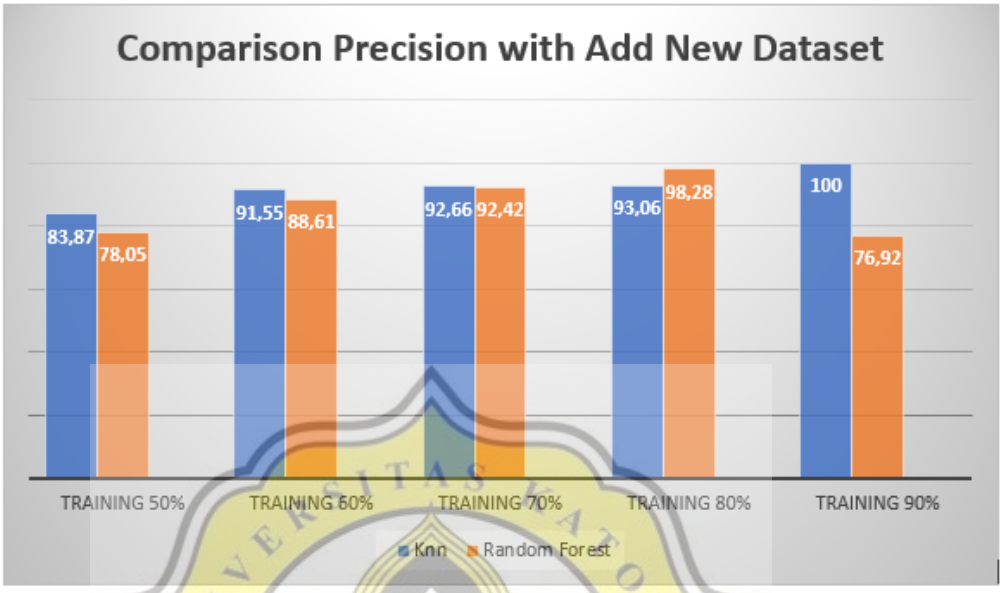

Illustration 19: Comparison Precision with Add New Dataset.

Based on Illustration 19, it shows that the results of the accuracy of the two algorithms by adding a new dataset. The addition of a new dataset aims to see whether the results of the previous precision with current precision have changed or not. If the precision results have a low value, there is a high probability of an error in the test being carried out. The smaller the False Positive, the greater the precision. From the results above, the KNN algorithm gets higher results than the split test data of 50%, 60%, 70%, 90% compared to the random forest algorithm. Meanwhile, the Random Forest Algorithm gets higher results in split testing by 80%. So it can be denied that the KNN Algorithm has fewer errors in the tests carried out than the Random Forest algorithm because the KNN Algorithm gets a higher value than the Random Forest Algorithm. So the addition of a new dataset

shows that the precision value of the two algorithms has increased due to the increased amount of data in search of a match.

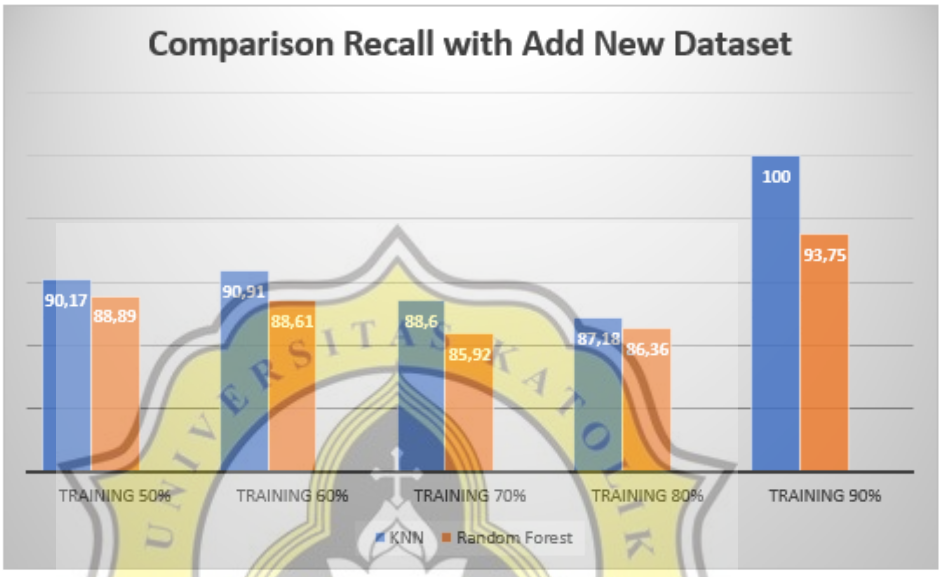

Illustration 20: Comparison Recall with Add New Dataset.

Illustration 20 shows that the recall result of both algorithms is by adding a new dataset. Recall value is looking for how to get the acquisition or amount. The greater the recall value does not guarantee a better precision value. From the results above, the KNN algorithm gets higher results from split data testing by 50%, 60%, 70%, 80%, 90% than the random forest algorithm. So it can be concluded that the KNN Algorithm has a better recall value than the Random Forest Algorithm because it has a higher recall value than the Random Forest algorithm.

The addition of a new dataset shows that the recall value of the KNN and random forest algorithms has increased and decreased. The split data test on the

KNN algorithm that has increased is the split data of 60% and 90%. Meanwhile, the Random Forest algorithm has an increase in recall value on split testing by 90%. The amount of recall value is influenced by the number of false negatives it has.

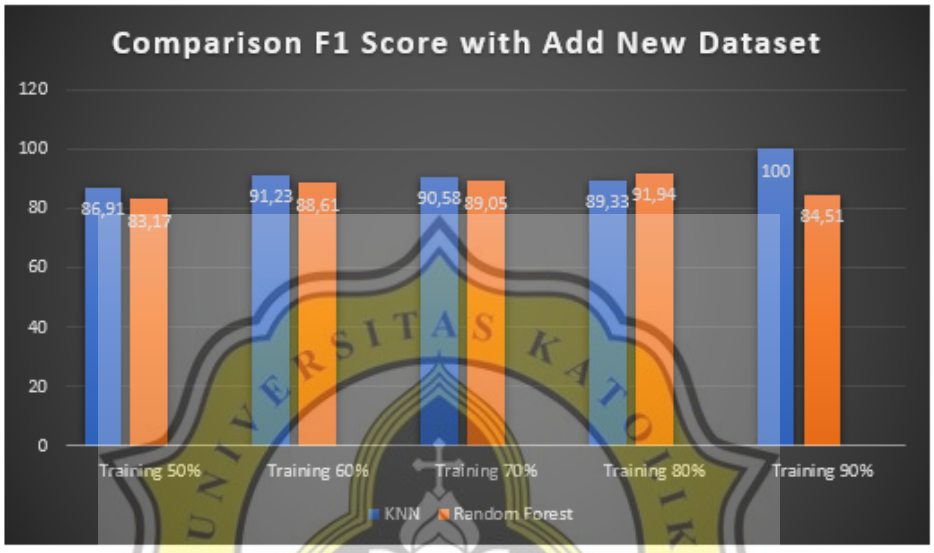

Illustration 21: Comparison F1 Score with Add New Dataset.

Based on illustration 21 it proves that the average precision and recall or f1 score of the KNN algorithm is better than the random forest algorithm on split data testing by 50%, 60%, 70%, and 90%. In contrast, the Random Forest Algorithm is better in f1 score on the split data test of 80%. The f1 score value is needed to find a balance between precision and recall. The addition of a new dataset to the f1 score shows that there is an increase in the value of the f1 score of the two algorithms both in the data split of 50%, 60%, 70%, 80% and 90%.