KENNESAW STATE
UNIVERSITY

CS 7267
MACHINE LEARNING

PROJECT 2

SUPERVISED LEARNING

INSTRUCTOR

Dr. Zongxing Xie

Michael Rizig
001008703

1. ABSTRACT

In this project, we are talked with implementing a Supervised Learning Algorithm, namely KNN, or K Nearest Neighbor. This algorithm is related to the Unsupervised Learning Algorithm K-Means as they both deal with clustering. The difference between KNN and K-Means is that KNN is a Supervised Algorithm, meaning that the data comes prelabeled and we can measure the accuracy and performance to later tune the model. First, we will discuss normalization, then we will normalize the data. After normalization we will partition the data between training and testing datasets (70-30 split respectively), and finally we will run our model with K=1,3,5,7, and 9 and measure each K values respective accuracy. After testing, we found that with our chosen dataset, our accuracy decreased as we increased K past 1. The highest accuracy recorded was 92.93% (k=1) and our lowest was 88.04% (k=9).
To view revision history and step by step building of this project view on my GitHub:
https://github.com/michaelrzg/Machine-Learning-Projects-Python

2. DATA NORMALIZATION

Our dataset is composed of 613 samples, each containing 29 datapoints and 1 label. In order to determine if our dataset requires normalization, we need to determine some simple statistics about our dataset. Below is a simple python function used to generate simple statistics.

```
def analyze(data):
    N = len(data)
    minValue = min(data)
    maxValue= max(data)
    dataRange = maxValue-minValue
    print("Total values: ",N,"\nMaximum value: ",maxValue,"\nMinimum Value: ",
minValue, "\nRange: ",dataRange)
```

When we run this function on our original pre-normalization data we get the following.

```
Total values:  18390
Maximum value:  4254.0
Minimum Value:  0.0
Range:  4254.0
```

Based on these statistics, we can observe that our data's range is very large (4254). This can cause inconsistency when we begin calculations as the scale of the larger numbers will effect the smaller numbers representation, and lead to skewed data. We need to normalize the data in order to identify a values true effect on the dataset and not its scaled effect. We normalize the data with the below function:
(each sample is a duple with its data in the first poison and label in second)

```
# normalize the dataset
def normalizeData(data):
    # for each sample we find its max and min elements and divide each point by the
range (max-min)
```

```
    for x in data:
        maxValue = max(x[0])
        minValue= min(x[0])
        for i in range (len(x[0])):
            x[0][i] = (x[0][i]-minValue)/(maxValue-minValue)
    return data
```

After running this function, all of our data will fall in the range of 0 and 1.
Our new statistics after normalization are the following:

```
Total values:  18390
Maximum value:  1.0
Minimum Value:  0.0
Range:  1.0
```

We can observe that after normalization, our range is now 1.

3. TEST RESULTS

2.1 Results with testing data

Runs for K = 1, 3, 5, 7, and 9.

Figure 2.1.a: K value: 1, Correct: 171, Error: 13, Accuracy: 92.93478260869566 %
Figure 2.1.b: K value: 3, Correct: 168, Error: 16, Accuracy: 91.30434782608695 %
Figure 2.1.c: K value: 5, Correct: 162, Error: 22, Accuracy: 88.04347826086956 %
Figure 2.1.d: K value: 7, Correct: 162, Error: 22, Accuracy: 88.04347826086956 %
Figure 2.1.e: K value: 9, Correct: 162, Error: 22, Accuracy: 88.04347826086956 %

Extra runs:

K value: 11 Correct: 160 Error: 24 Accuracy: 86.95652173913044 %
K value: 13 Correct: 160 Error: 24 Accuracy: 86.95652173913044 %
K value: 15 Correct: 159 Error: 25 Accuracy: 86.41304347826086 %
K value: 17 Correct: 160 Error: 24 Accuracy: 86.95652173913044 %

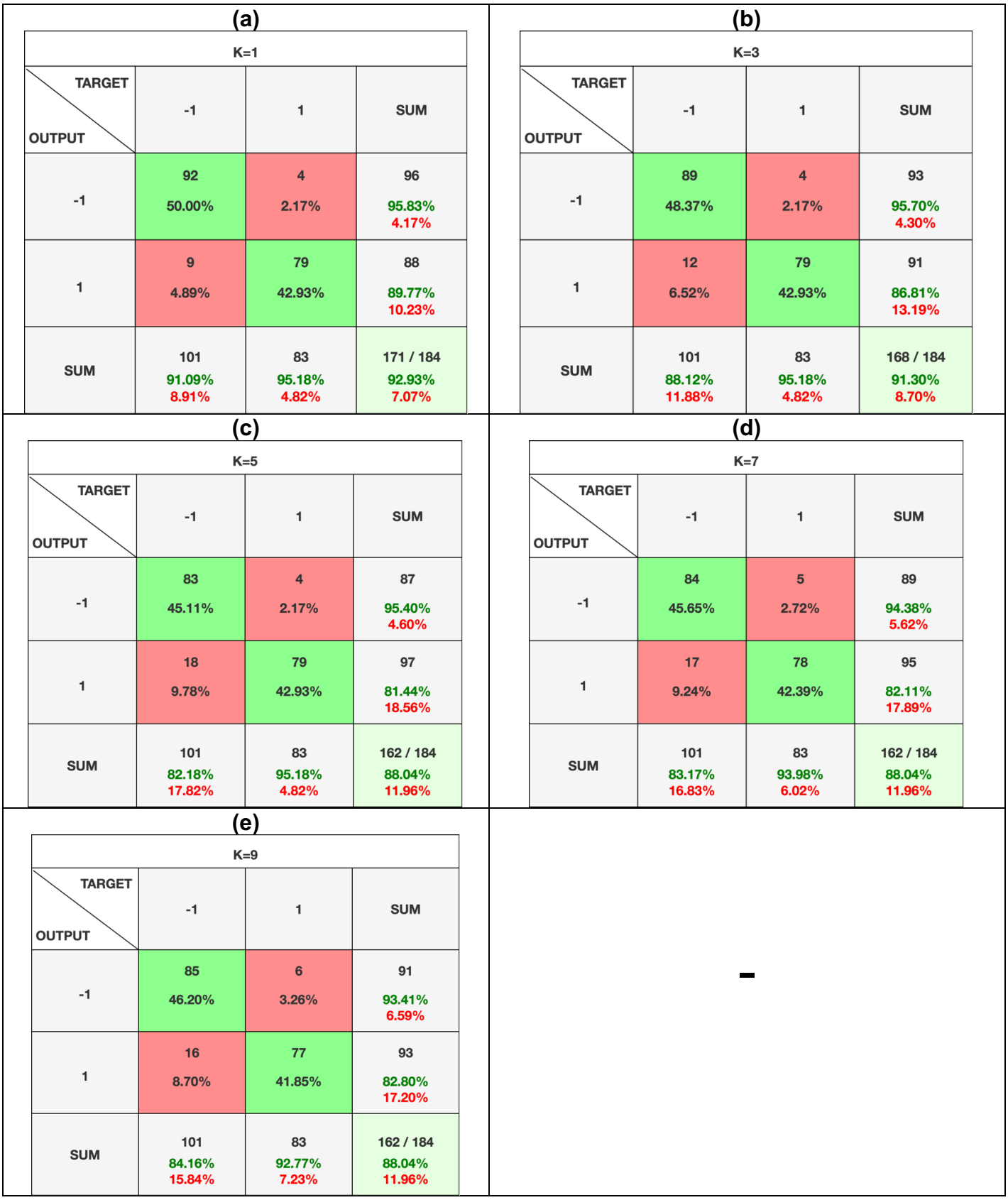

Figure 2.1: (a) K=1, **(b)** K=3, **(c)** K=5, **(d)** K=7, **(e)** K=9,

4.Discussion

As we can see from the test results, as we increased our number of K values, our accuracy decreased. This can be caused by many reasons, but for our dataset it seems our clusters were close together, meaning when we collect more than 1 datapoint we risk grabbing points from other clusters that are very nearby. Our highest observed accuracy was 92.93% with K=1 and our lowest accuracy observed within our original bounds of k=1,3,5,7,9 was 88.04% at K=9. Our overall lowest was observed when we go beyond our k bounds with 86.86 observed with K=19. Given more time, I would like to explore applying this concept to much larger datasets and seeing how our accuracy changes. By having a much larger dataset our model may run a little slower but will likely be able to accomplish more.

5. CODE

4.1 Code for k-nearest-neighbor.py

```
# Michael Rizig
# CS7247 Machine Learning
# Professor Zongxing Xie
# 9/24/24
# Assignment 2: K Nearest Neighbor

# import plot and math tools
import matplotlib.pyplot as plot
import seaborn as sea
import statistics
import math

class distanceCalculationModule:
    def __init__():
        pass
    # this function assigns a group to each input datapoint
    # data format:
    # data = [x1,x2...x613]
    # Datapoint x ∈ Data = ([x1,x2,..x29], label)
    # input format = [x1,x2...x29]
    # output format = int
    def predict(self,input, k, data):
        # define list to store distances from input to all datapoints
        distances = []
        # for each datapoint in sample input, calculate its distance to all other
points
        for datapoint in data:
            sum=0
            for i in range(len(input)-1):
                sum += (input[i]-datapoint[0][i])**2
            distances.append((math.sqrt(sum),datapoint))
```

```
        # sort list
        distances.sort(key=lambda x: x[0])
        # get k closest values
        kclosest = distances[:k]
        #debug
        #print("\n\ninput:",input, "closest:",kclosest)
        #tallay votes
        votes = []
        for value in kclosest:
            votes.append(value[1][1])
        # return the label with the highest votes (mode of set)
        return statistics.mode(votes)
    def run(self,inputSet, trainingSet):
        # predetermined k values to run
        kvalues = [1,3,5,7,9,11,13,15,17,19]
        for k in kvalues:
            # tally for confusion matrix later
            correct=0
            error=0
            tr=0
            bl=0
            n1=0
            p1=0
            #for each datapoint in the input set:
            # 1: remove label from sample array in first positon of duple,
            # 2: run predict to get predicted class for datapoint
            # 3: compare it to actual class (second position in duple)
            # 4: tally results
            for datapoint in inputSet:
                # remove last value ()
                datapoint[0].pop()
                # predict
                prediction = self.predict(datapoint[0],k,trainingSet)
                # compare and tally
                if(prediction != datapoint[1]):
                    if(int(prediction)==1):
                        tr+=1
                    else:
                        bl+=1
                    error+=1
                else:
                    correct+=1
                if(int(prediction) == -1):
                    n1+=1
                else:
                    p1+=1
            print("K value: ", k , " Correct: " , correct, " Error: ", error, "
Accuracy: " , (correct/(correct+error))*100 , "%")
```

```
            #print("top right: " , tr, " bottom left: ",bl, " Negative 1: ", n1, "
Positive 1: " ,p1)


# read in data from file to memory
def readData(path):
    # open file in read only
    file = open(path,'r',encoding='utf-8-sig')
    # create place to store each full line
    lines = []
    #parse each full line
    for line in file:
        lines.append(line[:-1])
    # create place to store each datapoint array
    output = []
    # for each full line, parse csv by spliting at ',',
    # store each datapoint in output with last value (label) sperated
    # output[x] = ([x1,x2...x29],label)
    for datapoint in lines:
        x = datapoint.split(",")
        # convert each array into an array of float values, and duple it with the
class label
        output.append(([float(x[i]) for i in range(len(x)-2)],x[len(x)-1]))
    return output

# normalize the dataset
def normalizeData(data):
    # for each sample we find its max and min elements and divide each point by the
range (max-min)
    for x in data:
        maxValue = max(x[0])
        minValue= min(x[0])
        for i in range (len(x[0])):
            x[0][i] = (x[0][i]-minValue)/(maxValue-minValue)
    return data

# normalize a single sample value
def normalizeInput(input):
    maxValue = max(input)
    minValue = min(input)
    output=[]
    for x in input:
        output.append((x-minValue)/(maxValue-minValue))
    return output

# Main: ------------------------------------------------------------------------
--------------------
```

```
# parse dataset:
dataset = normalizeData(readData("Data/wdbc.data.mb.csv"))

# define training and testing set ratio:
trainingSetCount = int(len(dataset) * .7)
testingSetCount = len(dataset)-trainingSetCount
# partition training set and testing set from full dataset:
trainingSet = dataset[:trainingSetCount]
testingSet = dataset[-testingSetCount:]
# run model
model = distanceCalculationModule()
model.run(testingSet,trainingSet)
```

4.2 Code for dataAnalytics.py

```
# Michael Rizig
# CS7247 Machine Learning
# Professor Zongxing Xie
# 9/24/24
# Assignment 2: K Nearest Neighbor - Data Analytics

# this file generates some simple statistics for the project report

import statistics
# read in data from file to memory
def readData(path):
    # open file in read only
    file = open(path,'r',encoding='utf-8-sig')
    # create place to store each full line
    lines = []
    #parse each full line
    for line in file:
        lines.append(line[:-1])
    # create place to store each datapoint array
    output = []
    # for each full line, parse csv by spliting at ',',
    # store each datapoint in output with last value (label) sperated
    # output[x] = ([x1,x2...x29],label)
    for datapoint in lines:
        x = datapoint.split(",")
        # convert each array into an array of float values, and duple it with the
class label
        for i in range(len(x)-1):
            output.append(float(x[i]))
    return output
def analyze(data):
    N = len(data)
```

```
    minValue = min(data)
    maxValue= max(data)
    dataRange = maxValue-minValue
    print("Total values: ",N,"\nMaximum value: ",maxValue,"\nMinimum Value: ",
minValue, "\nRange: ",dataRange)
# normalize the dataset
def normalizeData(data):
    # for each sample we find its max and min elements and divide each point by the
range (max-min)
    Maximum = max(data)
    Minimum = min(data)
    output = []
    for x in data:
        output.append((x-Minimum)/(Maximum-Minimum))
    return output
data = readData("Data/wdbc.data.mb.csv")
analyze(data)
norm =normalizeData(data)
analyze(norm)
```