CSE 5160 Project Report

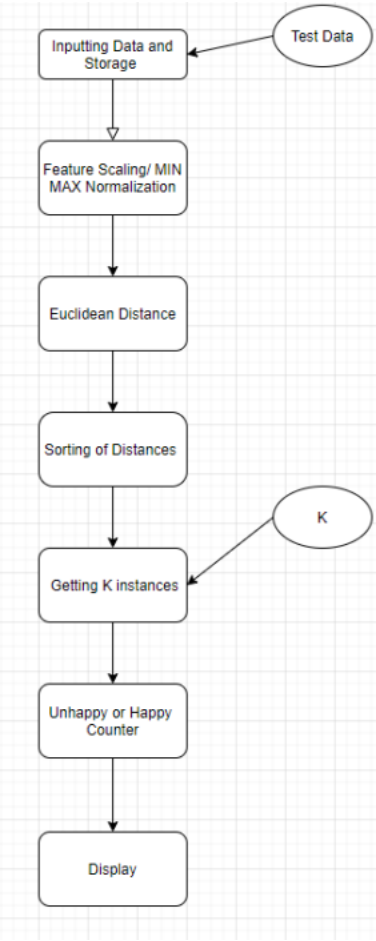
Jacob Diba *
College of Computer Science and Engineering
California State University,San Bernardino
San Bernardino CA 92407
csusb.edu/cse

Abstract

This report will examine the process of the creation of the KNN classifier, from the input of data to the output of the prediction from the algorithm. Then we will look at the type of data that was used and how it was used to build the KNN classifier. Also included is the testing of the KNN classifier from how it's tested to its accuracy.

*Use footnote for providing further information about author (webpage, alternative address)—*not* for acknowledging funding agencies.

1 Implementation Process of KNN Classifier



Flowchart of KNN Classifier for CSE 5160 Project

1.1 Inputting Data and Storage

The first part of the KNN Classifier is to take in all the data and set up a data structure to access said data later on. The data was set up in .csv file type which is a Comma-separated values file type that allows files to be stored tabular with commas separating the values. The first step was to accept the csv library into Python and clean the data so that there are no null or strange characters in the data. The next step is to input the data into an array. After we have completed that step a dictionary file structure which is a key/value pair type. After setting up the data the array then is inputted to the dictionary file structure and labels are created. This will then be repeated using a for a loop until all test date is located in the dictionary file structure

```python
def readcsv(fn):
    data_dict = dict()
    data_arr = []
    with open(fn) as d:
        # Cleaning of data due to Null Charcters
        # data = d.read()
        # data = data.replace('\x00', '')
        # data = data.replace('ÿþ', '')
        # data =data.replace("\n\n", "\n")
        # file = open('cleandata.csv', 'w')
        # file.write(data)
        # file.close()
        reader = csv.reader(d)
        flag = True

        for i in reader:
            data_dict=dict()
            if flag:
                flag = not flag
                continue
            #Dictonary of all given attrubuites
            data_dict["happy"]=int(i[0])
            data_dict["cityserv"]=int(i[1])
            data_dict["housecost"]=int(i[2])
            data_dict["schgual"]=int(i[3])
            data_dict["poltrust"]=int(i[4])
            data_dict["streets"]=int(i[5])
            data_dict["social"]=int(i[6])

            data_arr.append(data_dict)

    return data_arr
```

Read Function

1.2 Feature Scaling/ MIN MAX Normalization

After all data has been put in and labeled the said data must go through MIN MAX Normalization so that the data is not skewed one way. The first step that is happening in this function is to make sure that are deciding attribute or the attribute that tells us if they are happy or not is skipped so that does not cause errors in our prediction.

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Min-Max Normalization Formula

Then using the Min-Max Normalization formula above we get our minimum and our maximum and our value. After getting all of our values we input them into the formula and we get a feature value scaled. From that point on we repeat that step over and over again using a for loop until all of the data has been normalized and put back into the dictionary.

```python
def minmax(mm):
    normarr = []
    for val in mm:
        normdict= dict()
        for key, value in val.items():
            if key == "happy":
                normdict[key]=value
            else:
                norm = (value - 1) / (5 - 1)
                normdict[key] = norm
        normarr.append(normdict)
    return normarr
```

Min-Max Function

1.3 Euclidean Distance and Sorting

When all data has been Min-Max normalized the next step is to find the Euclidean distance of one test instance to all of the training instances.

$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

Euclidean Distance Formula Using the Euclidean Distance Formula from above input the test instance and the first training instance into the formula to find out that test instance's Euclidean Distance. After finding out the Euclidean Distance of that test instance put the value of that distance into a separate column in the dictionary data structure from before. This step will be repeated multiple times until all of the training data has been tested with the test instance and put into the distance label in the dictionary data structure.

When all training instances have been checked against the test instance to find the Euclidean Distance of all of them and inputted in. The next step is to sort the distance column from the shortest distance to greatest distance while still retaining all of the attributes that are associated with it.

These steps will occur for all testing instances that are in the test data.

```
def distance(data,instance):
    distancearr = []
    for item in data:  #distance formula
        s=0
        for k,v in item.items():
            if k == "happy" or k=="distance":
                continue
            s += math.pow(v-instance[k],2)
        dis = math.sqrt(s)
        item['distance']=dis
        distancearr.append(item)
```

```
def sort(data):
    sortedarr = sorted(data, key=lambda d: d['distance'])

    return sortedarr
```

Distance and Sort function

1.4 K Instances and Happy/Unhappy Counter

Once all training instances have added a distance to their distance column from the Euclidean Distance function. The next step is to get the K that the user has inputted or if the user has not inputted any K it will default 15 as a safety function built-in. Once a K has been established the function will select the number of K instances from the dictionary that has been sorted. Then have a counter set up to look at the decision attribute and count how many happy values there are or unhappy. Once that step has been completed the algorithm will decide on the result based on if the happy or unhappy counter is greater.

```
try:
    k = sys.argv[2]
except:
    k = 15

for instance in trainorm:
    happycount = 0
    unhappycount = 0
    distance(datanorm,instance)
    datanorm = sort(datanorm)
    for i in range(1,k+1):
        if datanorm[i]["happy"] == 1:
            happycount += 1
        else:
            unhappycount += 1
    if happycount > unhappycount:
        print('Instance: Happy')
        instance["perdicted value"] = 1
    elif happycount < unhappycount:
        print('Instance: Unhappy')
        instance["perdicted value"] = 0
```

K and Counter function

1.5 Display

Once all of the steps have been completed and the decision of the algorithm has been made the result will be displayed to the user on the prediction of the algorithm. If the algorithm came up with Happy then the output will come out as happy or vice versa if the encounter had come out as unhappy then the display will be unhappy. This step will then be repeated until all testing instances have been predicted and displayed to the user.

2 Data

2.1 Location of Data

The data that was chosen was from the University of California, Irvine Machine Learning Data Base.

Which can be found at

`http://archive.ics.uci.edu/ml/datasets/Somerville+Happiness+Survey`

2.2 Data Meaning

The data that was chosen is the Sommervile Happiness Survey which is a survey that was taken in Somervile Tennessee in 2018 that sees if a resident is happy or unhappy based on select attributes.

Those attributes being

- D = decision attribute (D) with values 0 (unhappy) and 1 (happy)
- X1 = the availability of information about the city services
- X2 = the cost of housing
- X3 = the overall quality of public schools
- X4 = your trust in the local police
- X5 = the maintenance of streets and sidewalks
- X6 = the availability of social community

Attributes X1 to X6 have values 1 to 5.

2.3 Data to Classifier

How the classifier is using the data that we have given it?

The first way that the Classifier is using the Data is to find the Euclidean Distance of every training instance to the instance that we are testing. This is by using the formula on the six different attributes together for that instance to find the distance. Then afterward getting the K number of instances and classifying the test instance.KNN however does not create the model but only does the prediction straight away. This is how KNN uses the data to classify the test instance that is given.

3 Classifier and Testing

3.1 Applying Classifier on test instances

How we applied the classifier was by taking 31 instances from our training data at random and then using them as test data. Then applying KNN to figure out if the said instance is Happy or Unhappy based on the attributes and distances. Then after the prediction has been given we look at the answer if the instance is happy or unhappy to see if the KNN classifier was correct or not.

3.2 Accuracy of KNN Classifier

The Accuracy of the KNN classifier was checked based on how many of the test instances it got correct and how many it got wrong while also changing the K value to see how it would affect the KNN Classifier.

K value	# of Instances	# Correct	# Incorrect	Accuracy	Test Error
5	31	23	8	74%	26%
15	31	19	12	60%	40%
20	31	19	12	60%	40%
30	31	20	11	64%	36%

4 Conclusion

In conclusion, when seeing the design of the KNN classifier the process can be seen. Starting at the input of the data then going on to min-max normalize our data. Afterward getting the Euclidean Distance and sorting those distances from the shortest to the largest. Then getting the K instances and counting the mood of those instances to let the algorithm then predict the mood of the test instance and display the outcome. The Data that was used for this Classifier can also be seen and the way that it was implemented to build our classifier. This can be seen with the number of attributes that we used and what each of them means. Lastly is the Accuracy of our KNN classifier which when tested seems to be the most accurate when having a K value of 5 with getting 74 percent accuracy rating.