

KNN: K-NEAREST NEIGHBOURS
CS456 - MACHINE LEARNING SPRING 2023

Rahul Vishwakarma, Jyothish Kumar J

School of Computer Sciences,
National Institute of Science Education and Research, Bhubaneswar,
Homi Bhabha National Institute

February 19, 2023

PART I: THEORY

1	Introduction	4
1.1	General Information	4
1.2	Special Points	6
2	Applications	7
3	Algorithm	8
3.1	Overview and Psuedocode	8
3.2	Distance Metrics	9

PART II: DEMONSTRATION

- 1 Data collection and processing 11**

- 2 Code 12**
 - 2.1 Image to Vector 12
 - 2.2 General classification scheme 13
 - 2.3 Handwriting recognition 14

- 3 Observations and Results 16**

Part I

THEORY

INTRODUCTION

GENERAL INFORMATION

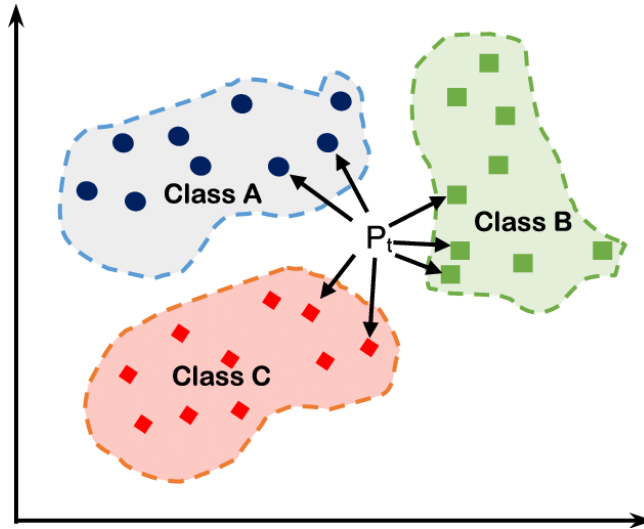


Figure. Visual representation of k-NN classification
[Gandhi n.d.]

- ▶ **k-NN** or **K-Nearest Neighbour** is a supervised classification algorithm.
- ▶ When a new piece of data is received, it's compared against all existing pieces of data for similarity. Once the top ' k ' nearest neighbors are identified, a majority vote is taken from these k data-points and the new point is assigned the majority vote as its class.
- ▶ Here ' k ' is the hyper-parameter responsible for controlling the inductive bias. [Harrington 2012]
- ▶ There is *no training involved* in this algorithm. These kinds of models are called Instance-based learning.

INTRODUCTION

GENERAL INFORMATION

- ▶ While chosen k can be both odd/even, Odd values of k is preferred since majority voting is done to get the classifying radius. If voting is near 50:50, weightage can be given to points to prevent anomalies.
- ▶ For large values of k , this model becomes computationally expensive.
- ▶ Computational geometry concepts such as **Voronoi diagrams** (Fig.2) are used for finding the neighborhood.

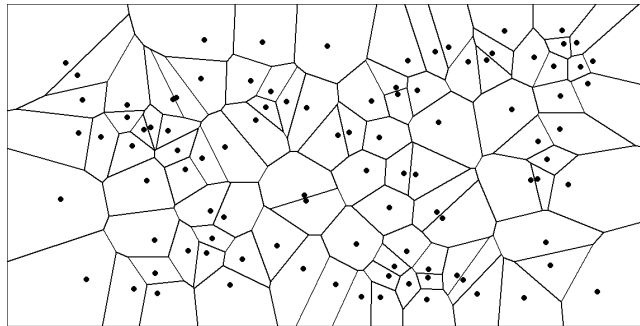


Figure. Voronoi Diagram [Bellelli n.d.]

INTRODUCTION

SPECIAL POINTS

- ▶ The algorithm has to carry around the full dataset; for large datasets, this implies a large amount of storage. In addition, you need to calculate the distance measurement for every piece of data in the database, and this can be cumbersome.
- ▶ An additional drawback is that kNN doesn't give you any idea of the underlying structure of the data; you have no idea what an "average" or "exemplar" instance from each class looks like.

- **Pros:** High accuracy, insensitive to outliers, no assumptions about data.
- **Cons:** Computationally expensive, requires a lot of memory.
- **Works with:** Numeric values and nominal values.

APPLICATIONS

The following example problem statements can be well addressed using kNN as a classifier.

1. **Handwriting recognition:** Given enough samples of handwritten specimen, a kNN classifier can be used to identify any new letter/number based on it's appearance similarity with the sample data. We will explore the implementation of this example in Part 2. [Code and Dataset obtained from [*GitHub - pbharrin/machinelearninginaction: Source Code for the book: Machine Learning in Action published by Manning — github.com* n.d.]]
2. **Match making on dating sites:** Classifier can match like-minded people using their inputs collected at the time of registration.
3. **Movie classification:** Classification of any given movie into genres such as romance, action, comedy etc. based on various features.

etc.

ALGORITHM

OVERVIEW AND PSEUDOCODE

After collection and preparation of data. The foundational steps involved in k-NN algorithm are as follows:

1. Distance calculation
2. Sorting dictionary
3. Voting with lowest k distances

Pseudocode is given as:

For every point in our dataset:

- calculate the distance between x and the current point
- sort the distances in increasing order
- take k items with lowest distances to x
- find the majority class among these items
- return the majority class as our prediction for the class of x

ALGORITHM

DISTANCE METRICS

- ▶ For the algorithm to work best on a particular dataset we need to choose the most appropriate distance metric accordingly. Some of the commonly used distance matrices for kNN are:

1. Euclidean Distance

$$d = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

2. Minkowski Distance

$$d = \left(\sum_{i=1}^n |x_i - y_i|^p \right)^{1/p}$$

3. Manhattan Distance

$$d = \sum_{i=1}^n |x_i - y_i|$$

Part II

DEMONSTRATION

DATA COLLECTION AND PROCESSING

- ▶ **Aim:** To design a classifier that recognizes a given image of a hand-written figure of a number between 0 to 9.
- ▶ **Data-set:** Over 2000 image samples of hand written numbers (0-9), approximately 200 samples per digit. Data made available in public domain by [Alpaydin and Kaynak n.d.].
- ▶ Obtained images are equivalent to a 32 x 32 matrix of 0s and 1s. Dark or inked areas of image reprinted by 1s and blank areas by 0s. [Fig. 3] These matrices are then converted to vectors of 1×1024 .¹

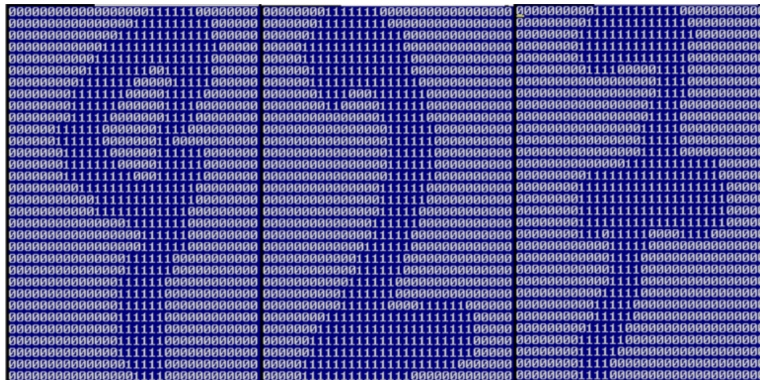


Figure. Binary representation of a sample image.

¹To make demonstration/replication easy, The converted vector has been provided for download [here.]

CODE

IMAGE TO VECTOR

The python code used to convert image files to 1×1024 vector on binary is given below:

```
1 def img2vector(filename):
2     returnVect = zeros((1,1024))
3     fr = open(filename)
4     for i in range(32):
5         lineStr = fr.readline()
6         for j in range(32):
7             returnVect[0,32*i+j] = int(lineStr[j])
8     return returnVect
```

Output:

```
>>> testVector = kNN.img2vector('testDigits/0_13.txt')
>>> testVector[0,0:31]
array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  1.,  1.,  1.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.])
>>> testVector[0,32:63]
array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,
        1.,  1.,  1.,  1.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.])
```

Figure. Output for the function `img2vector()`.

CODE

GENERAL CLASSIFICATION SCHEME

The below function (`classify0()`) is used to classify any data-set by calling the function with these 4 parameters:

1. Test vector
2. Training matrix
3. List of labels
4. K value (Hyperparameter)

```
1 def classify0(inX, dataSet, labels, k):
2     dataSetSize = dataSet.shape[0]
3     diffMat = tile(inX, (dataSetSize,1)) - dataSet
4     sqDiffMat = diffMat**2
5     sqDistances = sqDiffMat.sum(axis=1)
6     distances = sqDistances**0.5
7     sortedDistIndicies = distances.argsort()
8     classCount={}
9     for i in range(k):
10        voteIlabel = labels[sortedDistIndicies[i]]
11        classCount[voteIlabel] = classCount.get(voteIlabel,0) + 1
12    sortedClassCount = sorted(classCount.iteritems(), key=operator.itemgetter(1), reverse
13    =True)
14    return sortedClassCount[0][0]
```

CODE

HANDWRITING RECOGNITION

- ▶ We have compiled the procured dataset into Test and Training folders [Download]. Though the Training folder contains 2000 data-points (each being 1024-entry Floating Point Vectors).
- ▶ The python code used for handwriting recognition from our dataset is given in the next slide.
- ▶ The code tests the recognition of 900 Samples given in the Test folder.
- ▶ `function handwritingClassTest () ;` is a self contained classifier that tests our classifier. The code does three things:
 1. Parses each test file using OS functions in python.
 2. uses the previously discussed `img2vector ()` function to convert the sample to vector.
 3. sends the vector through the classification function (`classify0 ()`) along with the training matrix, lables and K value. to obtain the output.
 4. evaluates the output for error and reports the results along with accuracy of recognition / classification. The output of this function is shown in Fig. 5

CODE

HANDWRITING RECOGNITION

```
1 def handwritingClassTest():
2     hwLabels = []
3     trainingFileList = listdir('trainingDigits')           #load the training set
4     m = len(trainingFileList)
5     trainingMat = zeros((m,1024))
6     for i in range(m):
7         fileNameStr = trainingFileList[i]
8         fileStr = fileNameStr.split('.')[0]               #take off .txt
9         classNumStr = int(fileNameStr.split('_')[0])
10        hwLabels.append(classNumStr)
11        trainingMat[i,:] = img2vector('trainingDigits/%s' % fileNameStr)
12    testFileList = listdir('testDigits')                   #iterate through the test set
13    errorCount = 0.0
14    mTest = len(testFileList)
15    for i in range(mTest):
16        fileNameStr = testFileList[i]
17        fileStr = fileNameStr.split('.')[0]               #take off .txt
18        classNumStr = int(fileNameStr.split('_')[0])
19        vectorUnderTest = img2vector('testDigits/%s' % fileNameStr)
20        classifierResult = classify0(vectorUnderTest, trainingMat, hwLabels, 3)
21        print "the classifier came back with: %d, the real answer is: %d" % (
22            classifierResult, classNumStr)
23        if (classifierResult != classNumStr): errorCount += 1.0
24    print "\nthe total number of errors is: %d" % errorCount
25    print "\nthe total error rate is: %f" % (errorCount/float(mTest))
```







OBSERVATIONS AND RESULTS

```
>>> kNN.handwritingClassTest()
the classifier came back with: 0, the real answer is: 0
the classifier came back with: 0, the real answer is: 0
.
.
the classifier came back with: 7, the real answer is: 7
the classifier came back with: 7, the real answer is: 7
the classifier came back with: 8, the real answer is: 8
the classifier came back with: 8, the real answer is: 8
the classifier came back with: 8, the real answer is: 8
the classifier came back with: 6, the real answer is: 8
.
.
the classifier came back with: 9, the real answer is: 9
the total number of errors is: 11
the total error rate is: 0.011628
```

Figure. Output of function `handwritingClassTest()`

- ▶ The error rate obtained in our experiment = 1.2 %
- ▶ For each 900 test cases we had to do 2000 distance calculations on a 1024-entry floating point vector. *Though easy to implement, This is resource extensive and inefficient.* Additionally our dataset (`.txt` file) was also 2 mb.

REFERENCES I

-  [Alpaydin, E and C Kaynak \(n.d.\). “Optical recognition of handwritten digits data set. UCI Machine Learning Repository \(1998\)”. In: URL `https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits` \(\).](https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits)
-  [Bellelli, Francesco \(n.d.\). *The fascinating world of Voronoi diagrams* — `towardsdatascience.com`. `https://towardsdatascience.com/the-fascinating-world-of-voronoi-diagrams-da8fc700fa1b`. \[Accessed 19-Feb-2023\].](https://towardsdatascience.com/the-fascinating-world-of-voronoi-diagrams-da8fc700fa1b)
-  [Gandhi, Sai Kumar \(n.d.\). *Finding out Optimum Neighbours \(n\) number in the KNN classification using Python* — `medium.com`. `https://medium.com/analytics-vidhya/finding-out-optimum-neighbours-n-number-in-the-knn-classification-using-python-9bdcfefff58c`. \[Accessed 19-Feb-2023\].](https://medium.com/analytics-vidhya/finding-out-optimum-neighbours-n-number-in-the-knn-classification-using-python-9bdcfefff58c)
-  [GitHub - `pbharrin/machinelearninginaction`: Source Code for the book: *Machine Learning in Action* published by Manning — `github.com` \(n.d.\). `https://github.com/pbharrin/machinelearninginaction.git`. \[Accessed 19-Feb-2023\].](https://github.com/pbharrin/machinelearninginaction)
-  [Harrington, Peter \(Apr. 2012\). *Machine Learning in Action*. en. London, England: Simon and Schuster.](#)