Classifying Remote Sensing Data with KNN

Amit Kumar
Hochschule Mittweid
MAW1-16

Wasif Ahmed
Hochschule Mittweida
MAW1-16

January 21, 2019

1 Introduction

Landsat program is an ongoing program since 1972 with millions of images through satellite till now, in order to make research on global changes and application in agriculture, cartography, geology, forestry, regional planning, surveillance and education.
In this document we construct a classification model for the Coloroda data set(mountainous region in Colorado) to classify untrained data, for this we have 15,000 labeled data and 3.4 million unlabeled data samples with 6 dimensions and 14 classes.

2 K-Nearest Neighbor

There are many machine learning algorithm which can be use in order to train our dataset but we have use **k-nearest neighbor(KNN)** algorithm which is one of the simplest supervised machine learning algorithm and is fundamental type of classification i.e. data points are classified based on how its neighbor are classified.
k $\in$ (1 ... n) in KNN is a parameter that refers to the number of nearest neighbors to include in the majority voting process.

3 How KNN works?

Consider a graph in Fig.1 with some random input training data set, let say that class.1(Black points) and class.2(Green points), suppose we have a new data point i.e (red point) to be predict from which class it belongs.
consider k=3 which means we select three point which are least **euclidean distance** to new point as shown in fig.1 and after calculating distance from each point we can say that our new data point belongs to class.2(green points),if k=9 we look 9 different point closest to new data point as shown in fig.1.

Euclidean distance between two points in the plane with coordinates (x,y)and (a,b) is given by:

$$\text{dist(d)} = \sqrt{(x-a)^2 + (y-b)^2}$$

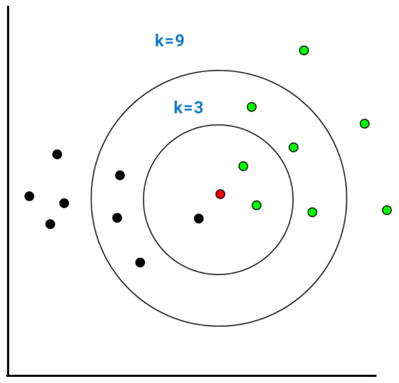

Figure 1: Graph

4 KNN Algorithm

Without more ado, lets see the following steps that how KNN can be execute in Python for a classification problem in Colorado dataset.

Steps:

- **Handle data** : Open the dataset from CSV and split it into test and training dataset
- **Similarity**: Calculate distance between two data instances
- **Neighbours**: Locate k most similar data instances
- **Response**: Generate a response from a set of data instances
- **Accuracy**: Calculate accuracy in test and training data

Algorithm:

First we load our CSV file by using open function to open file and to read the data line use reader function as our data is without and header line or any code so we display it as comma ',' separated values.

Algorithm 1: KNN Algorithm

```
import csv
with open(r'C:\Users\123\TrainingSamplesLabeled.csv') as csvfile:
lines =csv.reader(csvfile)
for row in lines:
print(','.join(row))
```

Now we split our data into training(80%) to make prediction and test(20%) to evaluate accuracy of the model, for this let use function loadDataSet which is using filename split ratio trainingset=[ ] and testset=[ ] as its input.

Algorithm 2: KNN Algorithm

```
import csv
import random
def loadDataSet(filename, split, trainingSet=[] , testSet=[]):
with open(filename, 'r') as csvfile:
lines = csv.reader(csvfile)
dataset = list(lines)
for x in range(len(dataset)-1):
for y in range(6):
dataset[x][y] = float(dataset[x][y])
if random.random() < split:
trainingSet.append(dataset[x])
else:
testSet.append(dataset[x])
```

In Algorithm 2 len(dataset) calculates the total number of data and for loop run tills the range of (lengths-1), range(6) is because we have 6 dimensional data and append(dataset[x]) define as soon 80%data is in training set,it adds the dataset in test data array.Now test the function:

Algorithm 3: KNN Algorithm

```
trainingset=[]
testset=[]
loadDataSet(r'C:\Users\123\TrainingSamplesLabeled.csv',0.80,
trainingset, testset)
print('train: ' +repr(len(trainingset)))
print('test: ' +repr(len(testset)))
```

```
train: 12096
test: 2903
```

So total number of training data set is 12096 and total number of test data set is 2903, which is finely split according to our defined function.

Now in order to make predictions calculate the similarity between any two giving data instances, this is needed to locate the k most similar data instances in the training dataset and in turn make a prediction, given that data measurements are in numeric and have same unit we use **euclidean distance**.

Algorithm 4: KNN Algorithm

```
import math
def euclideanDistance(instance1, instance2, length):
distance = 0
for x in range(length):
distance+= pow((instance1[x] - instance2[x]), 2)
return math.sqrt(distance)
```

In Algirithm 4 instance1 and instance2 are the two points of which euclidean distance is to be calculated and length denotes that how many attributes to involve.

For **k nearest neighbours** define a fuction getNeighbors which will return the k most similar neighbors from the training set for a giving test instance.

Algorithm 5: KNN Algorithm

```
import operator
def getNeighbors(trainingSet, testInstance, k):
distances = []
length = len(testInstance)-1
for x in range(len(trainingSet)):
dist= euclideanDistance(testInstance, trainingSet[x], length)
distances.append((trainingSet[x], dist))
distances.sort(key=operator.itemgetter(1))
neighbors= []
for x in range(k):
neighbors.append(distances[x][0])
return neighbors
```

In Algorithm 5 getNeighbors takes trainingSet, testInstance and k as input where k is number of nearest neighbor to be checked for.

To predict the response based on the neighbors in Algorithm 5 let each neighbor to vote for there class attributes and take the majority vote as a prediction i.e.

Algorithm 6: KNN Algorithm

```
import operator
def getResponse(neighbors):
classVotes = {}
for x in range(len(neighbors)):
response = neighbors[x][-1]
if response in classVotes:
```

```
classVotes[response] += 1
else:
classVotes[response] = 1
sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1),
reverse=True)
return sortedVotes[0][0]
```

We use getResponse which takes neighbors(output of getNeighbor fuction) as input.

So far we have defined all the function requires for the KNN algorithm, now to evaluate the accuracy of predictions, simplest way is to calculate the ratio of total correct prediction to all the predictions made.

Algorithm 7: KNN Algorithm

```
def getAccuracy(testSet, predictions):
correct= 0
for x in range(len(testSet)):
if testSet[x][-1] == predictions[x]:
correct += 1
return (correct/float(len(testSet))) * 100.0
```

Take getAccuracy function, place testSet and predictions inside.

Now compile all functions in one single main():

Algorithm 8: KNN Algorithm

```
def main():
#preparing data
trainingSet=[]
testSet=[]
split=0.80
loadDataSet(r'C:\Users\123\TrainingSamplesLabeled.csv', split,
trainingSet, testSet)
print('Train '+repr(len(trainingSet)))
print('Test '+repr(len(testSet)))
predictions=[]
k=17
for x in range(len(testSet)):
neighbors = getNeighbors(trainingSet, testSet[x], k)
result = getResponse(neighbors)
predictions.append(result)
print('> Predicted=' + repr(result)+ ',
actual=' +repr(testSet[x][-1]))
accuracy = getAccuracy(testSet, predictions)
print('Accuracy: ' + repr(accuracy) + '%')
main()
```

Here is the main() function with split value of 0.80 and K=17, let's run the main(): fuction:

Algorithm 9: KNN Algorithm

```
Train 12010
Test 2989
> Predicted='1', actual='1'
> Predicted='8', actual='8'
> Predicted='13', actual='13'
>      .              .
>      .              .
>      .              .
> Predicted='2', actual='4'
> Predicted='13', actual='13'
> Predicted='1', actual='1'
> Predicted='13', actual='13'
        Accuracy: 93.04115088658415%
```

In output of main(): fuction we have train dataset i which there are 12010 values and test data contains 2989 values with pridictive and actuals value and in total we got **Accuracy: 93.04115088658415%** which seem to be really good.