Fakulteit Ingenieurswese, Bou-omgewing & IT

Faculty of Engineering, Built Environment & IT

EAI 320: Artificial Intelligence

Imtiaz Mukadam

U13083113

Assignment 7: k-d Neighbours and Linear Regression

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Introduction

Statistical learning methods are used to classify discrete datasets or model continuous functions that can predict the output of a continuous dataset. In this practical, the nearest neighbor algorithm is investigated to classify a 3 dimensional dataset. Thereafter, a continuous function is modelled using linear regression on a 2 feature dataset.

Methodology

Nearest Neighbor algorithm

The nearest neighbor algorithm is implemented in the class KNN, which accepts 4 CSV files as object parameters. In this practical, the training and testing data files are given in 4 separate data files. The object contains various local variables that stores results and maintains the algorithms functionality.

The data is first sorted and stored as local variables within the KNN object.

```
def storeData(self):
    self.storeTrainingData()
    self.storeSampleData()

def storeTrainingData(self):
    for row in self.trainingDataSetFile:
        for i in range(len(row)):
            row[i] = float(row[i].strip())
        self.trainingDataSet.append(row)
    for row in self.trainingLabelsFile:
        for i in range(len(row)):
            row[i] = int(row[i])
        self.trainingLabels.append(row[0])  # there is only 1 label

def storeSampleData(self):
    for row in self.sampleSetFile:
        for i in range(len(row)):
            row[i] = float(row[i].strip())
        self.sampleSet.append(row)
    for row in self.sampleLabelsFile:
        for i in range(len(row)):
            row[i] = int(row[i])
        self.sampleLabels.append(row[0])
```

Figure 1 KNN data sorting

Thereafter, the function kNN() can be called to run the nearest neighbor algorithm that iterates through the algorithm and increments the k value until the error between the results and test samples reaches 0.

```
def kNN(self, k=1):
    error = 'Starting error is never 0'
    while error != 0:
        result = []
        for sample in self.sampleSet:
            result.append(self._kNN(sample, k=k))

        error = self.getError(result)
        self.printResults(result, error, k)
        k += 1

def _kNN(self, sample, k=1):
    distancePriorityQueue = Queue.PriorityQueue()
    sampleNumber = 0
    for test in self.trainingDataSet:
        distancePriorityQueue.put((self.getEuclideanDistance(sample, test), self.trainingLabels[sampleNumber]))
        sampleNumber += 1
    return self.bestNeighbor(distancePriorityQueue, k)

def getEuclideanDistance(self, sample, test):
    return math.sqrt((sample[0]-test[0])**2 +
                     (sample[1]-test[1])**2 +
                     (sample[2]-test[2])**2 +
                     (sample[3]-test[3])**2)

def bestNeighbor(self, neighbors, k):
    k_closest_neighbors = []
    for i in range(k):
        k_closest_neighbors.append(neighbors.queue[i][1])
    frequencyCounter = np.bincount(np.array(k_closest_neighbors))
    return np.argmax(frequencyCounter)  # return the most frequent neighbor
```

Figure 2 kNN() function that implements the nearest neighbor algorithm

The results of the algorithm is printed on screen and a plot of the sample and test set can be viewed by calling the objects plot() function.

Linear Regression

Similar to the nearest neighbor approach, the linear regression algorithm is implemented in its own LinearRegression object. The object accepts parameters for the dataset file path, the learning rate η, and the threshold error to terminate the algorithm.

The data is first extracted from the CSV files and sored as local variables.

```
class LinearRegression:

    def __init__(self, data_set='eyesightData/signdist.data', learning_rate=0.00001, threshold=0.001):
        self.dataSetFile = csv.reader(open(data_set))
        self.sampleAge = []
        self.sampleDistance = []
        if self.dataSetFile is not None:
            self.store_data()
        self.slope = 0
        self.intercept = 0
        self.learningRate = learning_rate
        self.threshold = threshold
        self.weights = None

    def store_data(self):
        for row in self.dataSetFile:
            self.sampleAge.append(float(row[0]))
            self.sampleDistance.append(float(row[1]))
```

The linear regression algorithm is run calling linear_regression(), which iterates through the algorithm, updating weights until the error is less than the threshold error.

```
def linear_regression(self):
    self.weights = self._linear_regression()
    self.plot()
    self.questions()

def _linear_regression(self):
    y_matrix = np.mat(self.sampleDistance).T
    x_matrix = np.mat([self.sampleAge,
                       [1]*len(self.sampleDistance)])
    weights = self.initialize_weights()
    error = 0.5*((y_matrix - x_matrix.T*weights).T*(y_matrix - x_matrix.T*weights))
    converged = False
    iterations = 0
    while not converged:
        iterations += 1
        gradient = x_matrix*(y_matrix - x_matrix.T*weights)
        weights += self.learningRate*gradient
        current_error = 0.5*((y_matrix - x_matrix.T*weights).T*(y_matrix - x_matrix.T*weights))
        print abs(current_error.item(0) - error.item(0))
        print iterations
        if abs(current_error - error) < self.threshold:
            return weights
        error = current_error
```

Figure 3 Method for linear regression

After the algorithm is complete, the objects plot() function is called to plot the dataset and resulting continuous function from the algorithm. Thereafter, questions() is called to produce output to answer the relevant questions of the practical.

Results

Nearest Neighbor results

Running kNN() from a KNN object produces the following results:

```
with k =   1
results =  [1, 2, 2, 2, 3, 1, 2, 3, 1, 3]
samples =  [1, 3, 3, 2, 3, 1, 2, 2, 1, 3]
Error =  3

with k =   2
results =  [1, 2, 2, 2, 3, 1, 2, 3, 1, 3]
samples =  [1, 3, 3, 2, 3, 1, 2, 2, 1, 3]
Error =  3

with k =   3
results =  [1, 3, 2, 2, 3, 1, 2, 3, 1, 3]
samples =  [1, 3, 3, 2, 3, 1, 2, 2, 1, 3]
Error =  2

with k =   4
results =  [1, 3, 2, 2, 3, 1, 2, 2, 1, 3]
samples =  [1, 3, 3, 2, 3, 1, 2, 2, 1, 3]
Error =  1

with k =   5
results =  [1, 3, 3, 2, 3, 1, 2, 3, 1, 3]
samples =  [1, 3, 3, 2, 3, 1, 2, 2, 1, 3]
Error =  1

with k =   6
results =  [1, 3, 2, 2, 3, 1, 2, 2, 1, 3]
samples =  [1, 3, 3, 2, 3, 1, 2, 2, 1, 3]
Error =  1

with k =   7
results =  [1, 3, 3, 2, 3, 1, 2, 2, 1, 3]
samples =  [1, 3, 3, 2, 3, 1, 2, 2, 1, 3]
Error =  0
```

Figure 4 Results of the kNN() algorithm

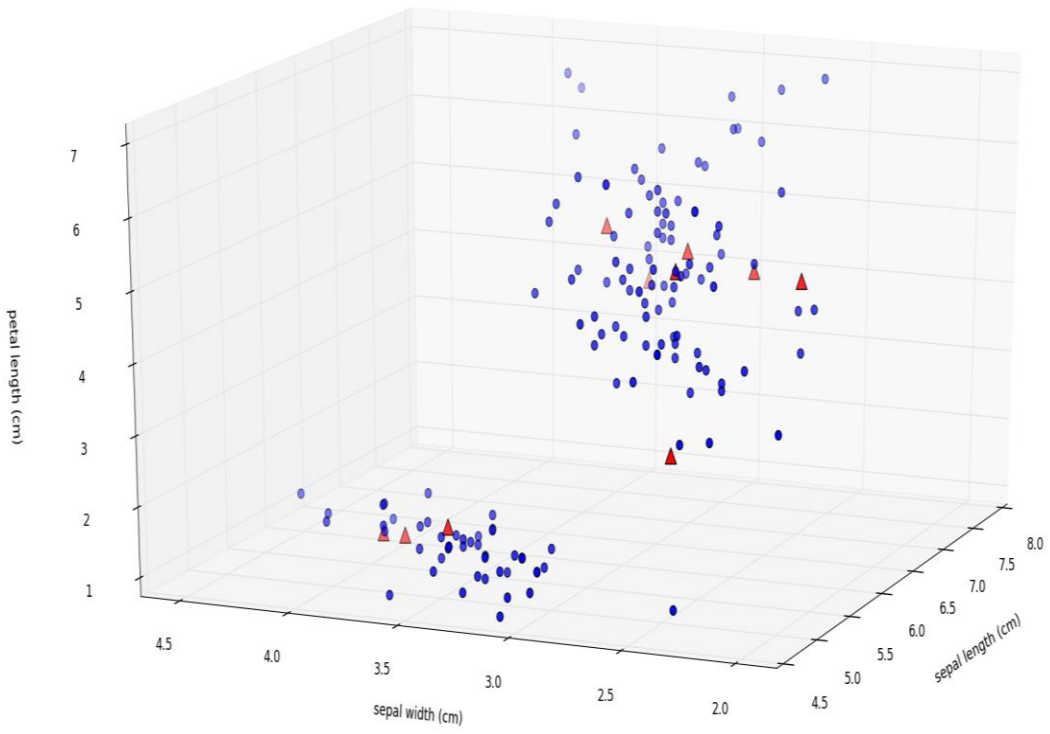


Figure 5 Dataset plot of the nearest neighbor question

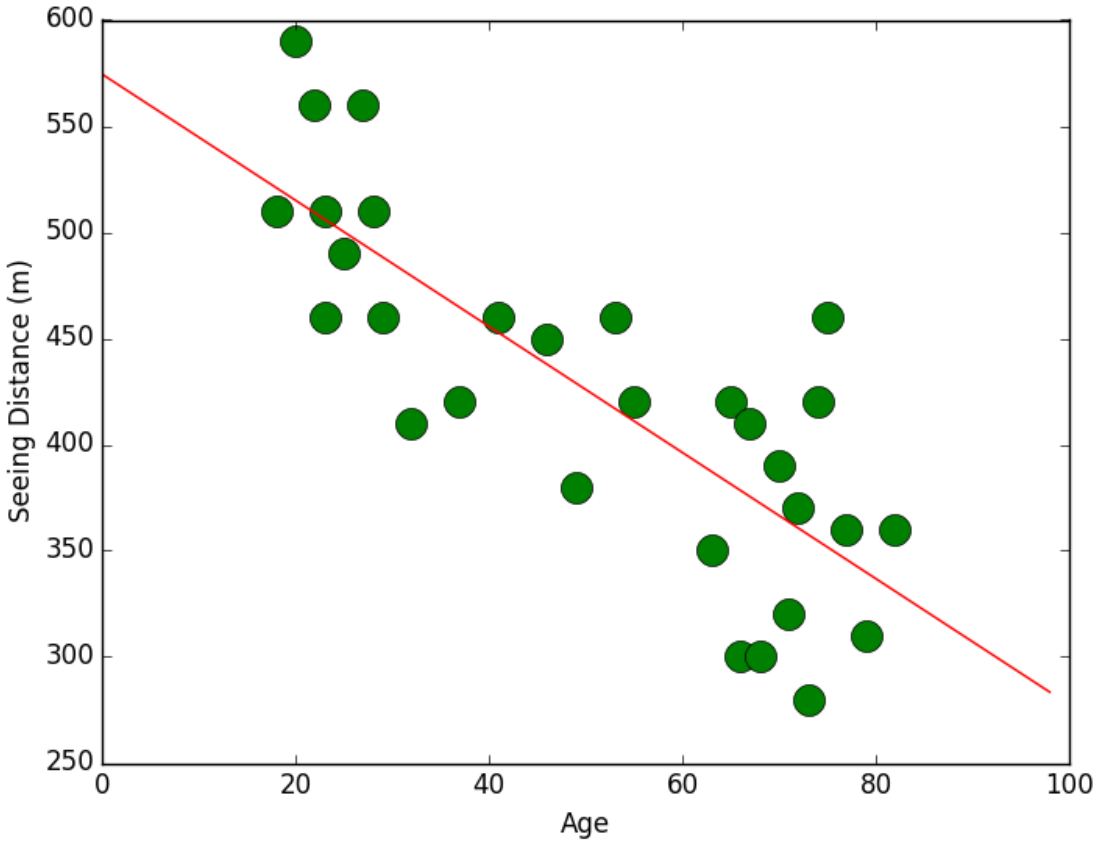
The dataset is plotted using the plot() function within the object. The red triangles represent the test set and the blue circles are considered to be its neighbors. The test data set is classified by considering its k closest neighbor.

Linear Regression

Calling the linear_regression() algorithm produces a continuous output of the iteration number, followed by the error difference.

With the learning rate as 0.00001 and a threshold of 0.001, the following results are achieved:

```
75660
0.00100000880047
75661
0.000999918898742
75662
Weight vector =  [[  -2.96974564]
 [ 574.4570987 ]]
Using the resulting function Y = wX + C :
The expected distance a 16 year old can see is  526.941168458
The expected distance a 90 year old can see is  307.179991092
```



Discussion

With a learning rate of 0.0001 i.e. 10^{-1} times more than the learning rate above, the linear regression algorithm diverges, thus the method continues indefinitely. It is for this reason an iteration limit is introduced. However, a learning rate that is too small will dramatically increase the time it takes for the threshold to be reached. Furthermore, using methods to guess the initial values of the weights can dramatically decrease the convergence time of the algorithm. In this practical, The weights are guessed by providing a linear approximation of the gradient and the closest value to the y axis as initial values to the weights vector.

The brute force approach was used for the nearest neighbor algorithm. For the small data set, the brute force approach proved adequate, however, intuitively it can be seen that the time complexity of the algorithm increases dramatically with larger, more practical datasets. Therefore, more power searching approaches such as the k-d tree approach should be employed to search and gather closest neighbors to a test sample.