homework sheet 02

Andre Seitz
03622870
andre.seitz@mytum.de

Linda Leidig
03608416
linda.leidig@tum.de

1 Assignment: Learning by doing

Problem 1 and 2

Python code required for the solution:

```
''' node.py '''

import numpy as np

class Node(object):
    '''
    class represents a node in a tree having children and a decision
    '''
    def __init__(self, features, values, current_depth, max_depth,
        classes):

        self.features = features
        self.values = values
        self.classes = classes

        # further slipping only if maximum depth is not reached and
            the data does not belong only to one class
        if current_depth < max_depth and np.unique(self.values[:,
            features.size-1]).size != 1:
            self.split_feature, self.split_value = self.find_decision
                ()
            data_leq, data_g = self.split_data(self.split_feature,
                self.split_value)
            self.child_leq = Node(self.features, data_leq,
                current_depth+1, max_depth, self.classes)    # node
                creates its children on its own
            self.child_g = Node(self.features, data_g, current_depth
                +1, max_depth, self.classes)

    # determining the feature and its value for splitting optimised by
         the gini index
    def find_decision(self):
        bins = np.append(self.classes, [self.classes.size])
        split_feature = ''
        split_value = -1000000000000000
        optimal_cost = 1000000000000000

        # try each feature and each value within
        for feature in range(0, self.features.size-1):
            for i in np.sort(self.values[:,feature]):
                current_data_leq, current_data_g = self.split_data(
                    feature, i)
```

```
            distribution_leq = np.histogram(current_data_leq[:,
                self.classes.size], bins)[0]
            distribution_g = np.histogram(current_data_g[:, self.
                classes.size], bins)[0]
            cost_leq = self.gini(distribution_leq)
            cost_g = self.gini(distribution_g)
            total = self.values[:,0].size
            current_cost = cost_leq*np.sum(distribution_leq)/total
                 + cost_g*np.sum(distribution_g)/total
            if current_cost < optimal_cost:
                optimal_cost = current_cost
                split_feature = feature
                split_value = i

    return split_feature, split_value


# split the data corresponding to a feature and its value
def split_data(self, split_feature, split_value):

    data_leq = self.values[self.values[:, split_feature]<=
        split_value]
    data_g = self.values[self.values[:, split_feature]>split_value
        ]

    return data_leq, data_g


# computes the gini index of a given distribution
def gini(self,v):
    sum_all = np.sum(v)
    sum_square = np.sum(np.square(v))
    if np.sum(v)==0:
        return 1.0
    else:
        return 1.0-(sum_square/((sum_all)**2.0))


# classifies a new data point
def classify(self, x):
    if hasattr(self, 'split_feature'):
        if x[self.split_feature] <= self.split_value:
            self.child_leq.classify(x)
        else:
            self.child_g.classify(x)
    else:
        bins = np.append(self.classes, [self.classes.size])
        distribution = np.histogram(self.values[:,self.classes.
            size], bins)[0]
        maximum = max(distribution)
        c = np.where(distribution == maximum)
        p = 1.0*maximum / np.sum(distribution)
        print(str(x) + ' belongs to class ' + str(c[0]) + ' with a
             probability of ' + str(p))


# visualizes the decision tree
def visualize(self, current_depth):
    if hasattr(self, 'split_feature'):
        bins = np.append(self.classes, [self.classes.size])
        distribution = np.histogram(self.values[:,self.classes.
            size], bins)[0]
        print (str(current_depth)  + '\t' + str(distribution)+ '\t
            ' + self.features[self.split_feature] + '<=' + str(
            self.split_value) +'\t'+ str(self.gini(distribution)))
```

```
            self.child_leq.visualize(current_depth+1)
            self.child_g.visualize(current_depth+1)
        else:
            bins = np.append(self.classes, [self.classes.size])
            distribution = np.histogram(self.values[:,self.classes.
                size], bins)[0]
            print (str(current_depth) + '\t'+str(distribution)+'\t\t'+
                str(self.gini(distribution)))
```

```
''' main.py '''

import sys
import numpy as np
import csv

import node


def main(argv):
    # read csv file
    csv_file = argv[1]
    with open(csv_file, 'r') as csvfile:
            reader = csv.reader(csvfile, delimiter=',')
            features = np.array(reader.next())
            values = np.array([row for row in reader], dtype = 'float'
                )

    classes = np.unique(values[:, features.size-1])     # vector with
        the specific classes

    root = node.Node(features, values, 0, 2, classes)   # create the
        root node, which creates its children on its own

    root.visualize(0)   # visualize the tree

    # Problem 2
    x1 = np.array([4.1, -0.1, 2.2])
    root.classify(x1)

    x2 = np.array([6.1, 0.4, 1.3])
    root.classify(x2)


if __name__ == '__main__':
    main(sys.argv)
```

Results:

1. Decision tree:

2. Classification:

[4.1 -0.1 2.2] belongs to class [1] with a probability of 1.0.

[6.1 0.4 1.3] belongs to class [2] with a probability of $\frac{1}{3}$.

Problem 3 and 4

Python code required for the solution:

```
''' kNN.py '''

import numpy as np
```

```
class KNN(object):
    '''
    classdocs
    '''

    def __init__(self, values):
        self.values = values

    # classifies a new data point
    def classify(self, x, k, classes):
        sorted_dist = self.find_rows_with_nn(x, k)
        rows = sorted_dist[0:k,0].astype(np.int32)

        cl = self.values[rows,self.values.shape[1]-1]

        bins = np.append(classes, [classes.size])
        distribution = np.histogram(cl, bins)[0]
        maximum = max(distribution)
        c = np.where(distribution == maximum)   #class
        p = 1.0*maximum / np.sum(distribution)  #probability

        return c[0], p

    # determines the class by regression
    def regress(self, x, k):
        sorted_dist = self.find_rows_with_nn(x, k)
        summe = 0
        normalization = 0
        for i in range(0, k):
            summe += 1/sorted_dist[i, 1]*self.values[sorted_dist[i,0],
                 self.values.shape[1]-1]
            normalization += 1/sorted_dist[i, 1]
        return summe/normalization

    # determines the rows in the values correpsonding to the k nearest
       neighbors
    def find_rows_with_nn(self, x, k):
        distances = np.zeros([self.values.shape[0],2])
        for i in range(0, self.values.shape[0]):
            j = self.values.shape[1]-1
            distances[i,1] = np.linalg.norm(x-self.values[i,0:j])
            distances[i,0] = i

        return np.sort(distances.view('i8,i8'), order=['f1'], axis=0).
           view(np.float)

''' main.py '''
import sys
import csv
import numpy as np

import kNN

def main(argv):
    # read csv file
    csv_file = argv[1]
    with open(csv_file, 'r') as csvfile:
        reader = csv.reader(csvfile, delimiter=',')
        features = np.array(reader.next())
        values = np.array([row for row in reader], dtype = 'float')

    classes = np.unique(values[:, features.size-1])     # vector with
        the specific classes

    knn = kNN.KNN(values)
```

```python
    k = 3

    # Problem 3
    x1 = np.array([4.1, -0.1, 2.2])
    c1, p1 = knn.classify(x1, k, classes)

    x2 = np.array([6.1, 0.4, 1.3])
    c2, p2 = knn.classify(x2, k, classes)

    print(str(x1) + ' belongs to class ' + str(c1) + ' with a
        probability of ' + str(p1))
    print(str(x2) + ' belongs to class ' + str(c2) + ' with a
        probability of ' + str(p2))


    # Problem 4
    reg1 = knn.regress(x1, k)
    reg2 = knn.regress(x2, k)

    print('Regression for '+ str(x1) + ' results in ' + str(reg1))
    print('Regression for '+ str(x2) + ' results in ' + str(reg2))


if __name__ == '__main__':
    main(sys.argv)
```

Results:

1. KNN:

[4.1 -0.1 2.2] belongs to class [0 1 2] with a probability of 0.333333333333 [6.1 0.4 1.3] belongs to class [2] with a probability of 0.666666666667

2. KNN Regression:

Regression for [4.1 -0.1 2.2] results in 0.561.

Regression for [6.1 0.4 1.3] results in 1.396.

Problem 5

The values of the second column are much smaller than the values of the first and third column. Scaling the data could compensate this problem affecting the euclidian distance (standardization).

This problem does not arise when training a decision tree since only one feature is considered for splitting the set at a time. Therefore, only the relative differences within each column are important.

2 Assignment: Probabilistic kNN

Problem 6

sdfd

Problem 7

sdfd

Problem 8

sdfd

3 Assignment: Neighbourhood Component Analysis

Problem 9

sdfd