Machine Learning Worksheet 2

Tomas Ladek
3602673
tom.ladek@tum.de

Problem 1

After parsing the data that was given in a csv file and making up an efficient data structure (matrix), the Gini index of the root node ($C = \{0, 1, 2\}$) was calculated:

$$i_G(t) = 1 - (\frac{5}{15})^2 - (\frac{6}{15})^2 - (\frac{4}{15})^2$$

Then in 0.1 steps from -0.6 to +10.0 (limits determined by data inspection), the Gini index of all possible left/right splits of the root node was calculated, for all features $x_{i,1}...x_{i,3}$. The maximum was a difference in Gini indices of ≈ 0.3615 for the first feature ($x_{i,1}$) for a split at the value 4.5[1]. The formula used for calculating the difference was

$$\Delta i_G(t)(x_{i,1} \leq s, t) = i_G(t) - \frac{\#classes\ in\ left\ tree}{\#classes\ in\ current\ data\ set} i_G(t_L) - \frac{\#classes\ in\ right\ tree}{\#classes\ in\ current\ data\ set} i_G(t_R)$$

with i_G being the Gini indices of the current node, the left tree and the right tree respectively.

Splitting the root node at $x_{i,1} = 4.5$ yielded a left tree (values less than or equal to the split value) consisting of a pure node (class distribution of 100% for class '1', Gini index 0) and a right tree combining the remaining classes '0' and '2', Gini index 0.4938. No further splits in the left tree were needed.

Once again performing a maximalization on the Gini index differences for every possible split of every feature in 0.1 steps yielded feature $x_{i,1}$ at 7.4 as the best split. The result was a left sub-tree with class distribution '2': $\frac{2}{3}$; '0': $\frac{1}{3}$ and a right sub-tree with a pure distribution of class '0'. The corresponding Gini indices were 0.4444 and 0 respectively.

As the maximum requested depth was reached, no further splitting was considered. The (raw) python code can be provided on request.

Problem 2

By using the tree that was constructed and explained above, the vector $\mathbf{x}_a$ is classified as '1' (following the left sub-tree, since $\mathbf{x}_{a,1} \leq 4.5$). Being a pure leaf node, $p(c = 1|\mathbf{x}_a, T) = 1$.

Vector $\mathbf{x}_b$ follows the right sub-tree ($\mathbf{x}_{b,1} > 4.5$) and ends up in its left leaf (($\mathbf{x}_{b,1} \leq 7.4$)). The corresponding classification is '2', because '2' is the majority class in that node. The probability of a correct classification is $p(c = 2|\mathbf{x}_b, T) = \frac{2}{3}$, due to the class distribution of that node.

[1] Due to the chosen calculation procedure, the split values take the maximum possible value between two different splits, instead of the average.

Problem 3

02_homework_knn

November 14, 2016

```
In [1]: import random
        import numpy as np
        import operator
        from sklearn import datasets
        import matplotlib.pyplot as plt
        %matplotlib inline
```

0.1 Load dataset

The iris data set (https://en.wikipedia.org/wiki/Iris_flower_data_set) it loaded by the function loadDataset.

Arguments:

- *split*: int: Split rate between test and training set e.g. 0.67 corresponds to 1/3 test and 2/3 validation

Returns:

- *X*: list(array of length 4); Trainig data
- *Z*: list(int); Training labels
- *XT*: list(array of length 4); Test data
- *ZT*: list(int); Test labels

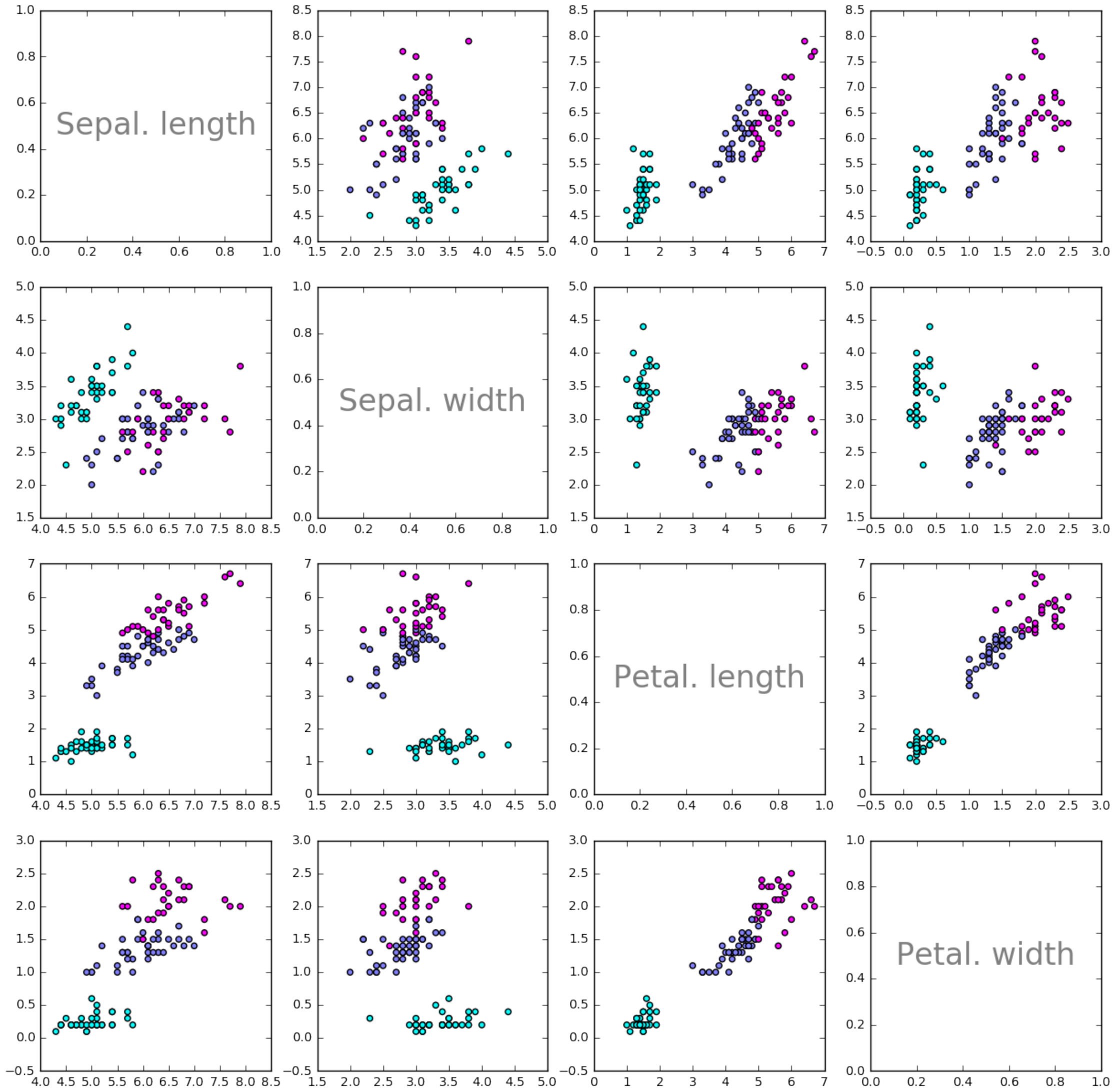
```
In [2]: def loadDataset(split, X=[] , XT=[], Z = [], ZT = []):
            dataset = datasets.load_iris()
            c = list(zip(dataset['data'], dataset['target']))
            random.seed(224)
            random.shuffle(c)
            x, t = zip(*c)
            sp = int(split*len(c))
            X = x[:sp]
            XT = x[sp:]
            Z = t[:sp]
            ZT = t[sp:]
            return X, XT, Z, ZT
```

```
In [3]: # prepare data
        split = 0.67
        X, XT, Z, ZT = loadDataset(split)
```

0.2 Plot dataset

Since X is dimentionality 4, 16 scatterplots (4x4) are plotted showing the dependencies of each two features.

```
In [4]: Xa = np.asarray(X)
        f, axes = plt.subplots(4, 4,figsize=(15, 15))
        for i in range(4):
            for j in range(4):
                if j == 0 and i == 0:
                    axes[i,j].text(0.5, 0.5, 'Sepal. length', ha='center', va='cent
                elif j == 1 and i == 1:
                    axes[i,j].text(0.5, 0.5, 'Sepal. width', ha='center', va='cente
                elif j == 2 and i == 2:
                    axes[i,j].text(0.5, 0.5, 'Petal. length', ha='center', va='cent
                elif j == 3 and i == 3:
                    axes[i,j].text(0.5, 0.5, 'Petal. width', ha='center', va='cente
                else:
                    axes[i,j].scatter(Xa[:,j],Xa[:,i], c = Z, cmap=plt.cm.cool)
```



0.3 Exercise 1: Euclidean distance

Compute euclidean distance between two data points.
arguments: * *x1*: array of length 4; data point * *x2*: array of length 4; data point
returns: * *distance*: float; euclidean distance between *x1* and *x2*

1 Implementation exercise: k-NN

```
In [5]: def euclideanDistance(x1, x2):
            return math.sqrt( pow((x1[0] - x2[0]), 2) + pow((x1[1] - x2[1]), 2) + p
```

1.1 Exercise 2: get k nearest neighbors

For one data point xt compute all k nearest neighbors.

arguments: * *X*: list(array of length 4); Trainig data * *Z*: list(int); Training labels * *xt*: array of length 4; Test data point

returns: * neighbors: list of length *k* of tuples (X_neighbor, Z_neighbor, distance between neighbor and xt); **this is the list of k nearest neighbors to xt**

```
In [6]: def getNeighbors(X, Z, xt, k):
            Xnp = np.array(X)
            datapoints = Xnp.shape[0]
            distance = np.empty(datapoints)
            for i in range(datapoints):
                distance[i] = euclideanDistance(Xnp[i], xt)
            nearest = distance.argsort()[:k]

            nearest_list.append((X[i], Z[i], distance[i]))
            return nearest_list
```

1.2 Exercise 3: get neighbor response

For the previously computed k nearest neighbors compute the actual response. I.e. give back the class of the majority of nearest neighbors. What do you do with a tie?

arguments: * neighbors: list((array, int, float) * c: int; number of classes in the dataset, for the iris dataset c=3

returns * y: int; majority target

```
In [7]: def getResponse(neighbors, c=3):
            classes = []
            labelCounts = list[0, 0, 0]
            for neighbor in neighbors:
                classes.append(neighbor[1])
            classes = list(set(classes))
            for neighbor in neighbors:
                if neighbor[1] == classes[0]:
                    labelCounts[0] = labelCounts[0] + 1
                elif neighbor[1] == classes[1]:
                    labelCounts[1] = labelCounts[1] + 1
                elif neighbor[1] == classes[2]:
                    labelCounts[2] = labelCounts[2] + 1
            classIndex = labelCounts.index(max(labelCounts))
            return classes[classIndex]
```

1.3 Exercise 4: Compute accuracy

arguments: * YT: list(int); predicted targets * ZT: list(int); actual targets

returns: * accuracy: float; percentage of correctly classified test data points

```
In [8]: def getAccuracy(YT, ZT):
            return 0
```

```
In [9]: def predict(X, Z, XT, k):
            Y=[]
            for xt in XT:
                neighbors = getNeighbors(X, Z, xt, k)
                Y.append(getResponse(neighbors))
            return Y
```

1.4 Testing

Should output an accuracy of 0.95999999999999996.

```
In [10]: # prepare data
         split = 0.67
         X, XT, Z, ZT = loadDataset(split)
         print 'Train set: ' + repr(len(X))
         print 'Test set: ' + repr(len(XT))
         # generate predictions
         k = 3
         YT = predict(X, Z, XT, k)
         accuracy = getAccuracy(YT, ZT)
         print('Accuracy: ' + repr(accuracy))
```

```
      File "<ipython-input-10-4285cf12047f>", line 4
        print 'Train set: ' + repr(len(X))
                          ^
    SyntaxError: invalid syntax
```

```
In [ ]:
```

Problem 4

Euclidean distance: $d = \sqrt{\sum_i (u_i - v_i)^2}$
Distances for $\mathbf{x}_a$:
$d_A = 2.76 \quad d_B = 3.78 \quad d_C = 2.18 \quad d_D = 5.97 \quad d_E = 3.22 \quad d_F = 3.97 \quad d_G = 2.93 \quad d_H = 2.83$
$d_I = 0.67 \quad d_J = 3.6 \quad d_K = 2.84 \quad d_L = 5.3 \quad d_M = 3.16 \quad d_N = 4.26 \quad d_O = 2.47$
$\Rightarrow$ 3-nearest neighbors are I, C, O corresponding to classes ('0', '2' and '1' respectively). Therefore:
$p(z = 0|x, 3) = \frac{1}{3} \cdot 1$
$p(z = 1|x, 3) = \frac{1}{3} \cdot 1$
$p(z = 2|x, 3) = \frac{1}{3} \cdot 1$

Distances for $\mathbf{x}_b$:
$d_A = 3.26 \quad d_B = 2.73 \quad d_C = 2.12 \quad d_D = 3.84 \quad d_E = 1.17 \quad d_F = 3.93 \quad d_G = 4.3 \quad d_H = 4.86$
$d_I = 1.75 \quad d_J = 5.7 \quad d_K = 3.15 \quad d_L = 3.77 \quad d_M = 5.32 \quad d_N = 6.45 \quad d_O = 4.56$
$\Rightarrow$ 3-nearest neighbors are E, I, C corresponding to classes ('2', '0' and '2' respectively). Therefore:
$p(z = 0|x, 3) = \frac{1}{3} \cdot 1 = \frac{1}{3}$
$p(z = 2|x, 3) = \frac{1}{3} \cdot 2 = \frac{2}{3}$

Problem 5

$$y = \frac{1}{\sum_{i \in N_k(x)} \frac{1}{d(x, x_i)}} \sum_{i \in N_k(x)} \frac{1}{d(x, x_i)} z_i$$

For $\mathbf{x}_a$: $y = 0.56$
For $\mathbf{x}_b$: $y = 1.398$