k-means-HVoe

August 27, 2023

0.0.1 Test a DHBW Seminarwork in W2022

Seminar task DW13 for DWH lecture in WS2022

[1]:
```
import pandas as pd # we need the dataframe structure from pandas
import numpy as np # import numpy for random, mean, sqrt function
from sklearn import datasets # import data
import matplotlib.pyplot as plt # import for plotting the data
import matplotlib.patches as mpatches # import for plotting the data
import copy # import function to get a deep copy of important values (otherwise␣
↪values could be overwritten)
%matplotlib inline


iris = datasets.load_iris() # load iris data set

# load array into pandas dataframe, name columns according to what value they␣
↪represent
initial_iris_data = pd.DataFrame(iris.data, columns=['Sepal Length', 'Sepal␣
↪Width', 'Petal Width', 'Petal Length']) # import the values from the iris␣
↪dataset and name the columns according to the values they represent
print(initial_iris_data)
# load which data set is connected to which species of iris
target = pd.DataFrame(iris.target, columns=['Target']) # load (scientific)␣
↪labeling of iris data


#check versions of libraries
print('pandas version is: {}'.format(pd.__version__))

print('numpy version is: {}'.format(np.__version__))

import sklearn
print('sklearn version is: {}'.format(sklearn.__version__))


# Import libary time to check execution date+time
import time
```

```python
# print the date & time of the notebook
print('*************************************************************************')
print("Actual date & time of the notebook:",time.strftime("%d.%m.%Y  %H:%M:%S"))
print('*************************************************************************')
```

```
     Sepal Length  Sepal Width  Petal Width  Petal Length
0             5.1          3.5          1.4           0.2
1             4.9          3.0          1.4           0.2
2             4.7          3.2          1.3           0.2
3             4.6          3.1          1.5           0.2
4             5.0          3.6          1.4           0.2
..            ...          ...          ...           ...
145           6.7          3.0          5.2           2.3
146           6.3          2.5          5.0           1.9
147           6.5          3.0          5.2           2.0
148           6.2          3.4          5.4           2.3
149           5.9          3.0          5.1           1.8

[150 rows x 4 columns]
pandas version is: 1.0.1
numpy version is: 1.18.1
sklearn version is: 0.22.1
***********************************************************************
Actual date & time of the notebook: 26.08.2023  20:04:40
************************************************************************
```

[2]:
```python
# Initialization: create 3 centroids (for 3 clusters)
def createRandomCentroids(k, seed):
    np.random.seed(seed = seed)
    centroids = {
        i+1: [
            initial_iris_data['Sepal Length'].min() + np.random.random() *␣
 ↪(initial_iris_data['Sepal Length'].max() - initial_iris_data['Sepal Length'].
 ↪min()),
            initial_iris_data['Sepal Width'].min() + np.random.random() *␣
 ↪(initial_iris_data['Sepal Width'].max() - initial_iris_data['Sepal Width'].
 ↪min()),
            initial_iris_data['Petal Width'].min() + np.random.random() *␣
 ↪(initial_iris_data['Petal Width'].max() - initial_iris_data['Petal Width'].
 ↪min()),
            initial_iris_data['Petal Length'].min() + np.random.random() *␣
 ↪(initial_iris_data['Petal Length'].max() - initial_iris_data['Petal Length'].
 ↪min())
        ]
```

```
        for i in range(k) # create an array of length k with index 1 to index k␣
↪entries which represent the cluster centroids. The entries themself are arrays␣
↪(in this case of length 4)
    }
    return centroids
```

[3]:
```
# draw initial cluster centers on two graphs
def drawInitialCentroids(data, centroids):
    plt.figure(figsize=(12, 3))
    colors = np.array(['red', 'green', 'blue'])

    # plot the datavalues sepal length in comparison to their sepal width
    plt.subplot(1, 2, 1)
    plt.scatter(data['Sepal Length'], data['Sepal Width'], c="black")
    for i in centroids.keys():
        plt.scatter(centroids[i][0], centroids[i][1], s=100, color=colors[i - 1])
    plt.title('Sepal Length vs Sepal Width')

    # plot the datavalues petal length in comparison to their petal width
    plt.subplot(1, 2, 2)
    plt.scatter(data['Petal Length'], data['Petal Width'], c="black")
    for i in centroids.keys():
        plt.scatter(centroids[i][3], centroids[i][2], s=100, color=colors[i - 1])
    plt.title('Petal Length vs Petal Width')
```

[4]:
```
def assignPointsToCluster(df, centroids):
    for i in centroids.keys():
        # calc euclidian distance for every row in pandas dataframe df
        df['dist_to_{}'.format(i)] = (
            np.sqrt(
                (df['Sepal Length'] - centroids[i][0]) ** 2
                + (df['Sepal Width'] - centroids[i][1]) ** 2
                + (df['Petal Width'] - centroids[i][2]) ** 2
                + (df['Petal Length'] - centroids[i][3]) ** 2
            )
        )
    # assign point to the centroid with smallest distance
    df['cluster'] = df[['dist_to_1','dist_to_2', 'dist_to_3']].idxmin(axis=1) #␣
↪search for the smallest distance along the calculated distances
    df['cluster'] = df['cluster'].map(lambda x: int(x.lstrip('dist_to_'))) #␣
↪remove "dist_to_" from the cluster string and convert it into int (to get the␣
↪data sets connected to cluster k)
```

[5]:
```
# draws the current state of the clustering. With the clustercenters beeing␣
↪alpha 1 and the points alpha 0.5
def drawFigure(data, centroids):
    plt.figure(figsize=(12,3))
```

```python
    colors = np.array(['red', 'green', 'blue'])
    iris_targets_legend = np.array(iris.target_names)
    red_patch = mpatches.Patch(color='red', label='Setosa')
    green_patch = mpatches.Patch(color='green', label='Versicolor')
    blue_patch = mpatches.Patch(color='blue', label='Virginica')

    plt.subplot(1, 2, 1)
    plt.scatter(data['Sepal Length'], data['Sepal Width'],␣
 ↪c=colors[data['cluster'] - 1], alpha = 0.5)
    for i in centroids.keys():
        plt.scatter(centroids[i][0], centroids[i][1], s = 100, color=colors[i -␣
 ↪1])
    plt.title('Sepal Length vs Sepal Width')
    plt.legend(handles=[red_patch, green_patch, blue_patch])

    plt.subplot(1,2,2)
    plt.scatter(data['Petal Length'], data['Petal Width'],␣
 ↪c=colors[data['cluster'] - 1], alpha = 0.5)
    for i in centroids.keys():
        plt.scatter(centroids[i][3], centroids[i][2], s= 100, color=colors[i -␣
 ↪1])
    plt.title('Petal Length vs Petal Width')
    plt.legend(handles=[red_patch, green_patch, blue_patch])
```

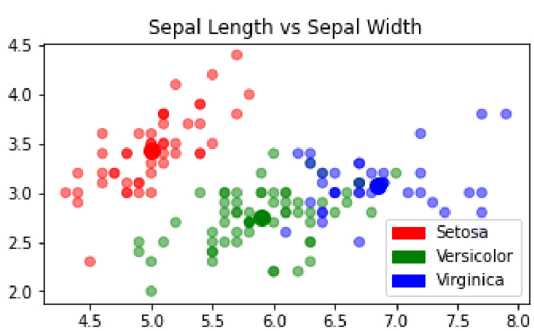
[6]:
```python
# calc new centroids using weight (using the mean function of each value, that␣
 ↪is assigned to the cluster)
def calcNewCentroids(df, centroids):
    for i in centroids.keys():
        centroids[i][0] = np.mean(df[df['cluster'] == i]['Sepal Length'])
        centroids[i][1] = np.mean(df[df['cluster'] == i]['Sepal Width'])
        centroids[i][2] = np.mean(df[df['cluster'] == i]['Petal Width'])
        centroids[i][3] = np.mean(df[df['cluster'] == i]['Petal Length'])
```

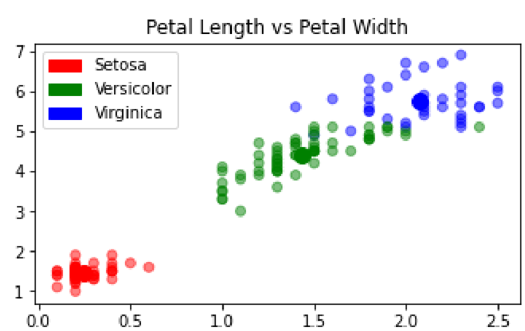
[7]:
```python
# compare the predicition of the clustering against the actual values and return␣
 ↪a success percentage
def calcAccuracy(prediction, target):
    correct = 0
    for i in prediction.index:
        if prediction[i] == target[i] + 1:
            correct += 1
    return correct / prediction.count()
```

[8]:
```python
# iterate until stable (until centroids do not change)
iterationCount = 0

def kMeans(data, numberOfClusters):
    bestAccuracy = 0
```

```
    bestPrediction = None
    bestCentroids = None

    for i in range(50):
        centroids = createRandomCentroids(numberOfClusters, i)
        while True:
            oldCentroids = copy.deepcopy(centroids) # copy old centroids (otherwise they would be overwritten)
            assignPointsToCluster(data, centroids)
            calcNewCentroids(data, centroids)
            # if the centroids have not changed in the current run of k-means that means that clustering is at the current best and will not change any further
            if centroids == oldCentroids:
                break
        accuracy = calcAccuracy(data['cluster'], target['Target']) # calculate accuracy of current run
        if accuracy > bestAccuracy:
            print(f"{i}: {accuracy}")
            # copy all the important values from the currently most sucessfull run for returning it at the end of the function
            bestPrediction = data.copy(deep = True)
            bestCentroids = copy.deepcopy(centroids)
            bestAccuracy = accuracy
    drawFigure(bestPrediction, bestCentroids) # draw the best clustering we can achive by using k-means
    return bestPrediction

bestPrediction = kMeans(initial_iris_data, 3) # run k means with iris data, with 3 clusters and save output of the function to bestPrediction
```

```
0: 0.24
1: 0.58
16: 0.6466666666666666
27: 0.6666666666666666
28: 0.8933333333333333
```





[9]:
```python
# create confusion matrix to analyse labeling

import sklearn.metrics as sm
from sklearn.metrics import ConfusionMatrixDisplay

sm.confusion_matrix(bestPrediction['cluster'], target['Target'])
```

[9]:
```
array([[ 0,  0,  0,  0],
       [50,  0,  0,  0],
       [ 0, 48, 14,  0],
       [ 0,  2, 36,  0]], dtype=int64)
```

[10]:
```python
# print current date and time

print("Date & Time:",time.strftime("%d.%m.%Y  %H:%M:%S"))
# end of import test
print ("*** End of K-means-HVoe Jupyter Notebook ***")
```

```
Date & Time: 26.08.2023  20:04:51
*** End of K-means-HVoe Jupyter Notebook ***
```