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

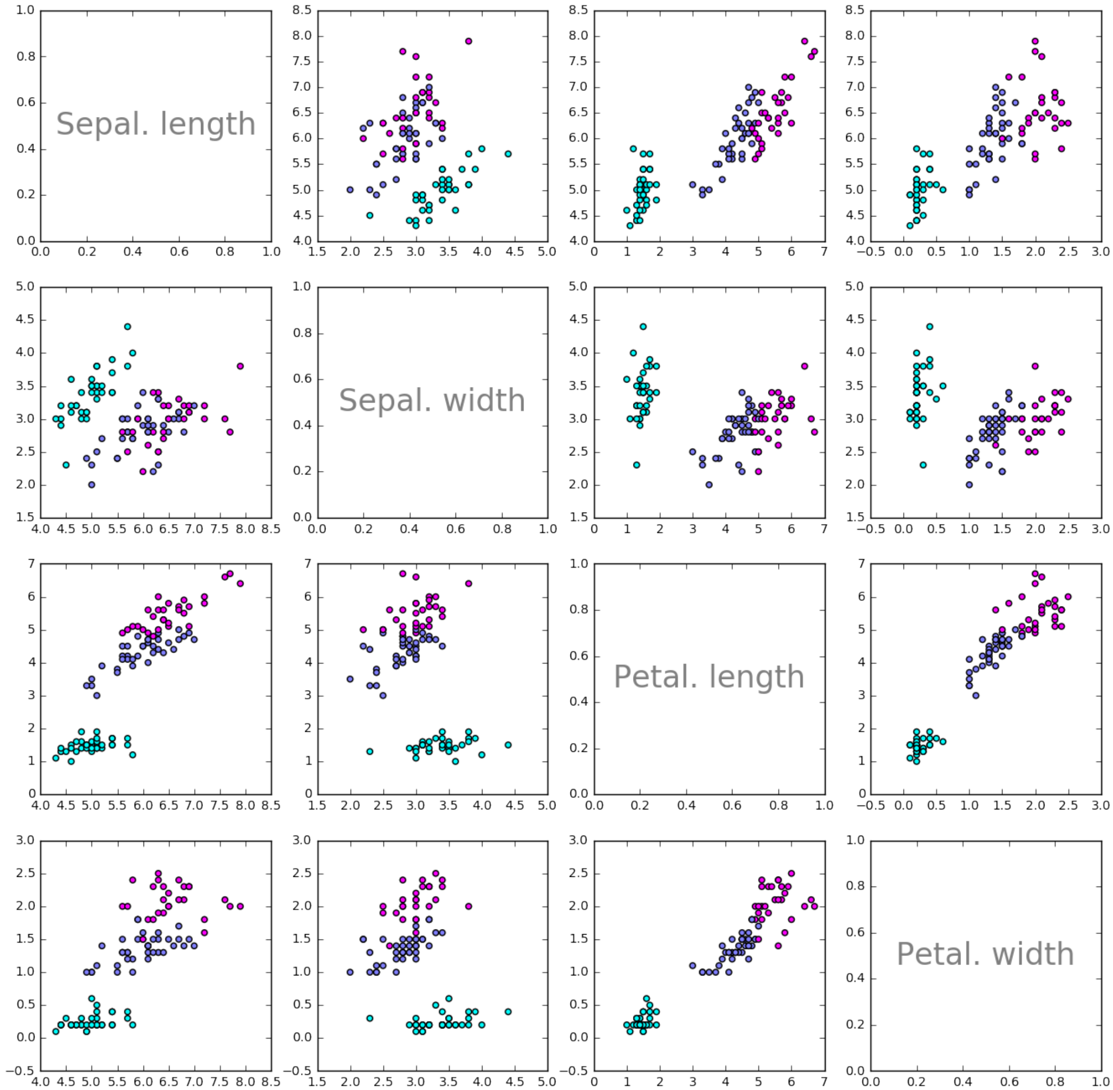
```
In [2]: def loadDataset(split, X=[] , XT=[], Z = [], ZT = []):
            dataset = datasets.load_iris()
            c = list(zip(dataset['data'], dataset['target']))
            random.seed(224)
            random.shuffle(c)
            x, t = zip(*c)
            sp = int(split*len(c))
       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