Machine Learning Worksheet Solution 1

Julius Jankowski

1 Decision Trees

Problem 1:

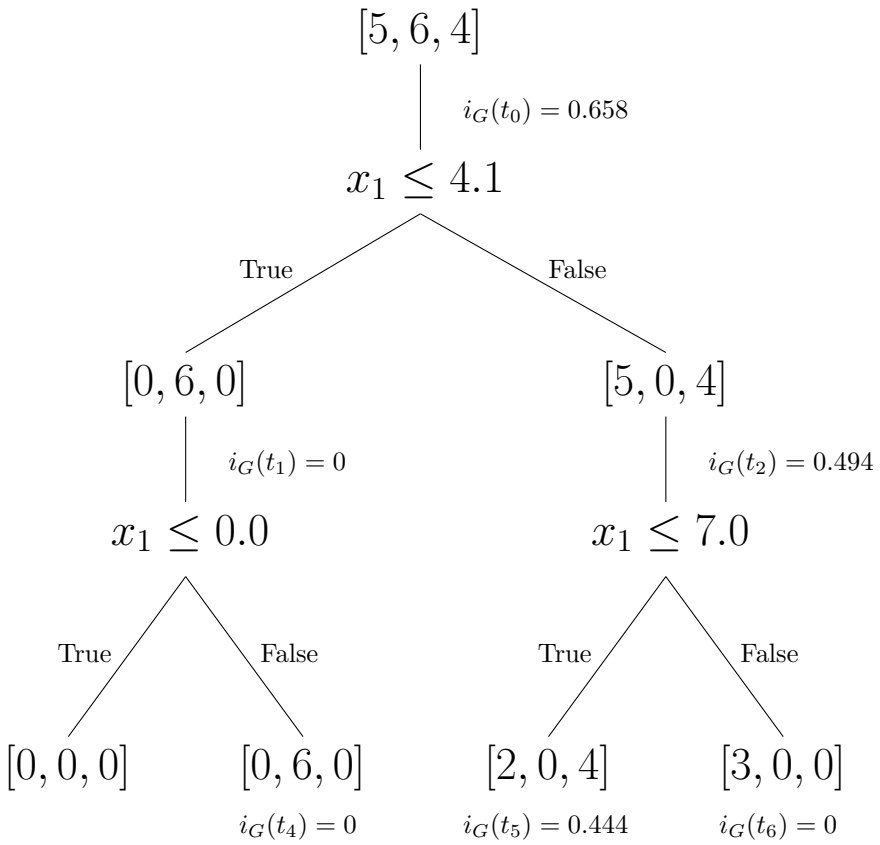


Figure 1.1: Decision Tree T. The stepsize between possible decision values was chosen to be 0.1.

Problem 2:

Classification of $\mathbf{x}_a = (4.1, -0.1, 2.2)^T$ based on Decision Tree T:

$$y_a = 1 \tag{1.1}$$

$$p(c = y_a \mid \mathbf{x}_a, T) = 1.0 \tag{1.2}$$

Classification of $\mathbf{x}_b = (6.1, 0.4, 1.3)^T$ based on Decision Tree T:

$$y_b = 2 \tag{1.3}$$

$$p(c = y_b \mid \mathbf{x}_b, T) = 0.67 \tag{1.4}$$

2 k-Nearest Neighbors

Problem 3: see last page.

Problem 4:

Classification of $\mathbf{x}_a = (4.1, -0.1, 2.2)^T$ based on 3-NN applied to the dataset:

$$y_a = 0 \tag{2.1}$$

Here, all 3 classes are among the 3 nearest neighbors. Thus the result is determined by the tie handling.

Classification of $\mathbf{x}_b = (6.1, 0.4, 1.3)^T$ based on 3-NN applied to the dataset:

$$y_b = 2 \tag{2.2}$$

Problem 5:

Regression of $\mathbf{x}_a = (4.1, -0.1, 2.2)^T$ based on 3-NN applied to the dataset:

$$\hat{y}_a = 0.561 \tag{2.3}$$

Regression of $\mathbf{x}_b = (6.1, 0.4, 1.3)^T$ based on 3-NN applied to the dataset:

$$\hat{y}_b = 1.396 \tag{2.4}$$

Problem 6:

The problem with the given dataset is that the three features are scaled differently. While the feature x_1 is distributed over a range of 9.5, the feature x_2 is distributed over a range of 1.6 and feature x_3 is distributed over a range of 4.5. Hence, differences in lower scaled features have a much lower impact on the euclidean distance than differences in wider distributed features. However, this problem does not come along when applying the decision tree since it just bases on a binary comparison (higher or lower than ..). This scaling effects can be tackled by normalizing the features based on the stochastic mean value and the standard deviation.

01_homework_knn

October 29, 2017

1 Programming assignment 1: k-Nearest Neighbors classification

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

1.1 Introduction

For those of you new to Python, there are lots of tutorials online, just pick whichever you like best :)

If you never worked with Numpy or Jupyter before, you can check out these guides * https://docs.scipy.org/doc/numpy-dev/user/quickstart.html * http://jupyter.readthedocs.io/en/latest/

1.2 Your task

In this notebook code to perform k-NN classification is provided. However, some functions are incomplete. Your task is to fill in the missing code and run the entire notebook.

In the beginning of every function there is docstring, which specifies the format of input and output. Write your code in a way that adheres to it. You may only use plain python and `numpy` functions (i.e. no scikit-learn classifiers).

Once you complete the assignments, export the entire notebook as PDF using nbconvert and attach it to your homework solutions. On a Linux machine you can simply use `pdfunite`, there are similar tools for other platforms too. You can only upload a single PDF file to Moodle.

1.3 Load dataset

The iris data set (https://en.wikipedia.org/wiki/Iris_flower_data_set) is loaded and split into train and test parts by the function `load_dataset`.

```
In [2]: def load_dataset(split):
            """Load and split the dataset into training and test parts.

            Parameters
            ----------
            split : float in range (0, 1)
                Fraction of the data used for training.
```

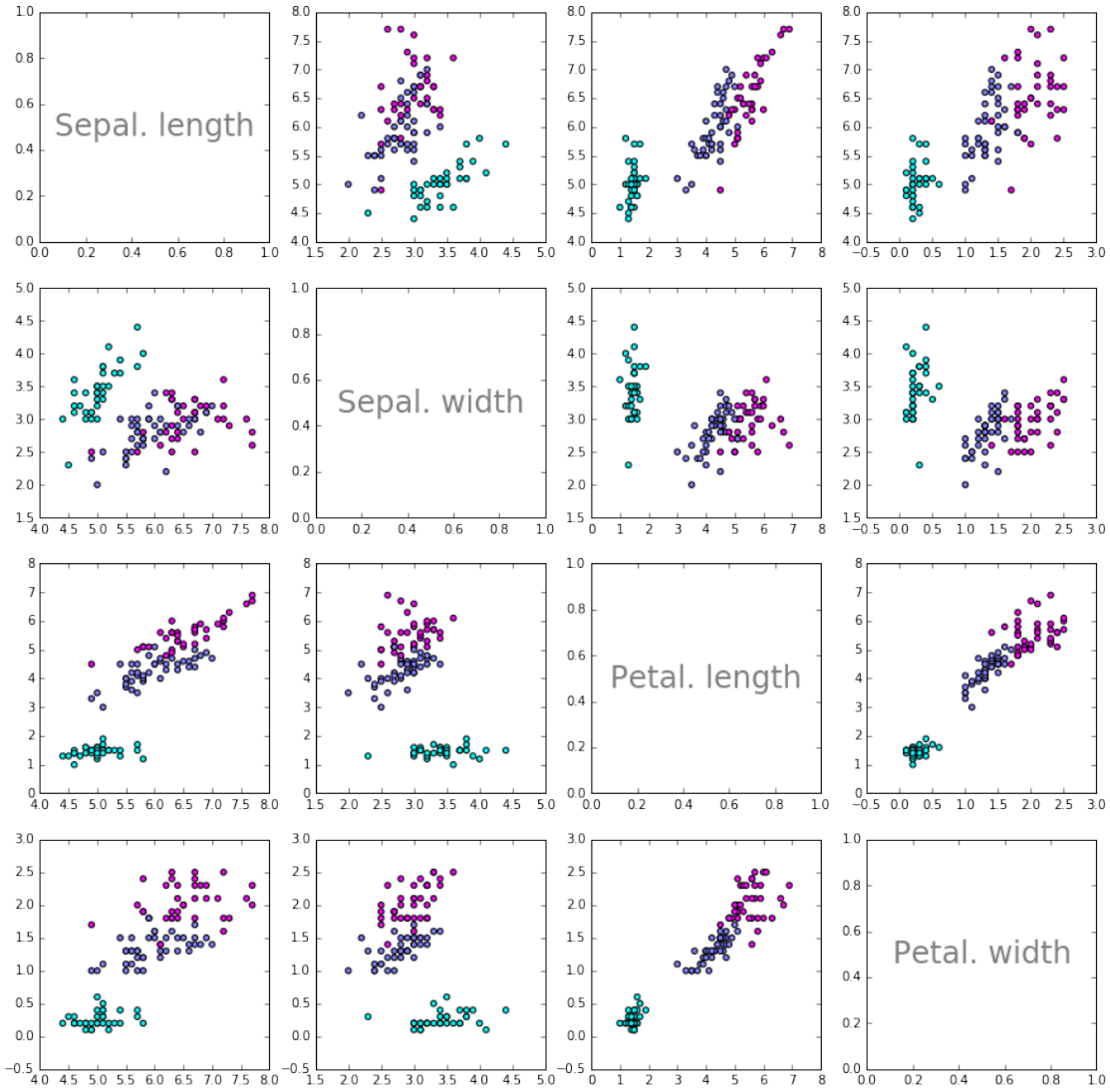
```
        Returns
        -------
        X_train : array, shape (N_train, 4)
            Training features.
        y_train : array, shape (N_train)
            Training labels.
        X_test : array, shape (N_test, 4)
            Test features.
        y_test : array, shape (N_test)
            Test labels.
        """
        dataset = datasets.load_iris()
        X, y = dataset['data'], dataset['target']
        X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, random_sta
        return X_train, X_test, y_train, y_test
```

```
In [3]: # prepare data
        split = 0.67
        X_train, X_test, y_train, y_test = load_dataset(split)
```

1.4 Plot dataset

Since the data has 4 features, 16 scatterplots (4x4) are plotted showing the dependencies between each pair of features.

```
In [4]: 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='center', size=24,
                elif j == 1 and i == 1:
                    axes[i,j].text(0.5, 0.5, 'Sepal. width', ha='center', va='center', size=24,
                elif j == 2 and i == 2:
                    axes[i,j].text(0.5, 0.5, 'Petal. length', ha='center', va='center', size=24,
                elif j == 3 and i == 3:
                    axes[i,j].text(0.5, 0.5, 'Petal. width', ha='center', va='center', size=24,
                else:
                    axes[i,j].scatter(X_train[:,j],X_train[:,i], c=y_train, cmap=plt.cm.cool)
```



1.5 Task 1: Euclidean distance

Compute Euclidean distance between two data points.

```
In [5]: def euclidean_distance(x1, x2):
            """Compute Euclidean distance between two data points.

            Parameters
            ----------
            x1 : array, shape (4)
                First data point.
            x2 : array, shape (4)
                Second data point.
```

```
    Returns
    -------
    distance : float
        Euclidean distance between x1 and x2.
    """

    d = np.linalg.norm(x1-x2)

    return d
```

1.6 Task 2: get k nearest neighbors' labels

Get the labels of the *k* nearest neighbors of the datapoint *x_new*.

```
In [30]: def get_neighbors_labels(X_train, y_train, x_new, k):
             """Get the labels of the k nearest neighbors of the datapoint x_new.

             Parameters
             ----------
             X_train : array, shape (N_train, 4)
                 Training features.
             y_train : array, shape (N_train)
                 Training labels.
             x_new : array, shape (4)
                 Data point for which the neighbors have to be found.
             k : int
                 Number of neighbors to return.

             Returns
             -------
             neighbors_labels : array, shape (k)
                 Array containing the labels of the k nearest neighbors.
             """

             N_train = X_train.shape[0]
             Distances = []

             # Build up a vector with the distances
             for x in X_train:
                 Distances.append(euclidean_distance(x_new, x))

             # Get the maximum distanced index
             max_idx = np.argmax(Distances)

             neighbors_idx = []
             neighbors_labels = []

             for j in range(k):
```

```
            min_dist  = Distances[max_idx]
            min_idx = max_idx
            for i in range(N_train):
                if Distances[i] < min_dist:
                    allready_assigned_flag = 0
                    for idx in neighbors_idx:
                        if i == idx:
                            allready_assigned_flag = 1
                            break
                    if allready_assigned_flag == 0:
                        min_dist = Distances[i]
                        min_idx = i

            neighbors_idx.append(min_idx)
            neighbors_labels.append(y_train[min_idx])

        return neighbors_labels
```

1.7 Task 3: get the majority label

For the previously computed labels of the k nearest neighbors, compute the actual response. I.e. give back the class of the majority of nearest neighbors. Think about how a tie is handled by your solution.

```
In [7]: def get_response(neighbors, num_classes=3):
            """Predict label given the set of neighbors.

            Parameters
            ----------
            neighbors_labels : array, shape (k)
                Array containing the labels of the k nearest neighbors.
            num_classes : int
                Number of classes in the dataset.

            Returns
            -------
            y : int
                Majority class among the neighbors.
            """

            class_votes = np.zeros(num_classes)

            for c in neighbors:
                class_votes[c] = class_votes[c] + 1

            # Get the class index with the highest number of votes. A tie is handled by the argm
            return np.argmax(class_votes)
```

1.8 Task 4: compute accuracy

Compute the accuracy of the generated predictions.

In [11]:
```
def compute_accuracy(y_pred, y_test):
    """Compute accuracy of prediction.

    Parameters
    ----------
    y_pred : array, shape (N_test)
        Predicted labels.
    y_test : array, shape (N_test)
        True labels.
    """

    N_test = len(y_pred)
    pos_num = 0

    for i in range(N_test):
        if y_pred[i] == y_test[i]:
            pos_num = pos_num + 1

    return pos_num/N_test
```

In [12]:
```
# This function is given, nothing to do here.
def predict(X_train, y_train, X_test, k):
    """Generate predictions for all points in the test set.

    Parameters
    ----------
    X_train : array, shape (N_train, 4)
        Training features.
    y_train : array, shape (N_train)
        Training labels.
    X_test : array, shape (N_test, 4)
        Test features.
    k : int
        Number of neighbors to consider.

    Returns
    -------
    y_pred : array, shape (N_test)
        Predictions for the test data.
    """
    y_pred = []
    for x_new in X_test:
        neighbors = get_neighbors_labels(X_train, y_train, x_new, k)
        y_pred.append(get_response(neighbors))
    return y_pred
```

1.9 Testing

Should output an accuracy of 0.9473684210526315.

```
In [31]: # prepare data
         split = 0.67
         X_train, X_test, y_train, y_test = load_dataset(split)
         print('Training set: {0} samples'.format(X_train.shape[0]))
         print('Test set: {0} samples'.format(X_test.shape[0]))

         # generate predictions
         k = 3
         y_pred = predict(X_train, y_train, X_test, k)
         accuracy = compute_accuracy(y_pred, y_test)
         print('Accuracy = {0}'.format(accuracy))
```

```
Training set: 112 samples
Test set: 38 samples
Accuracy = 0.9473684210526315
```