Exercise

01

TUM Department of Informatics

Supervised by Prof. Dr. Stephan Günnemann
Informatics 3 - Professorship of Data Mining and Analytics

Submitted by Marcel Bruckner (03674122)
Julian Hohenadel (03673879)
Kevin Bein (03707775)

Submission date Munich, October 25, 2019

kNN Classification

Problem 1:

a)

L1-Norm: $||x||_1 = \sum_{i=1}^{n} |x_i|$

L1-Distance: $d_1(x_1, x_2) = |x_1 - x_2| = \sum_{i=1}^{n} |x_{1,i} - x_{2,i}|$

$d_1(A,B) = 1.5 \quad d_1(B,A) = 1.5 \quad d_1(C,A) = 1.5$

$d_1(D,E) = 2.5 \quad d_1(E,F) = 1 \quad d_1(F,E) = 1$

b)

L2-Norm: $||x||_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}$

L2-Distance: $d_2(x_1, x_2) = \sqrt{|x_1 - x_2|^2} = \sqrt{\sum_{n=1}^{n} |x_{1,i} - x_{2,i}|^2}$

$d_2(A,B) \approx 1.12 \quad d_2(B,A) \approx 1.12 \quad d_2(C,A) \approx 1.5$

$d_2(D,C) \approx 2.2 \quad d_2(E,F) = 1 \quad d_2(F,E) = 1$

c) The nearest neighbors are the same but for point D. Here, the L1-Distance is smallest for point E whereas with L2-Distance, the nearest point is C. Both have different advantages but in a 2D setting, L2 represents the closest point better (as can be visually confirmed).

Problem 2:

a) x_new will be always be classified as as C because when we set k to be equal to the number of all points, the class with most points will win. In this case, it is C with $64 > 32 > 16$.

b) It is not possible to make a clear statement here since the classification is based on the weights and these are unknown. Any class is possible depending on the kernel chosen.

Problem 3:

False, the feature test performed in a node is done by only evaluating one feature to split the training data. Therefore decision boundaries can only be parallel to the axis respectivly. This means that a decision tree at depth 1 can never produce a diagonal as shown in the plot. For that reason a huge depth would be needed to approximate the diagonal in a zig-zag like pattern.

Problem 4:

$i_H(t) = -\sum_{c_i \in C} \pi_{c_i} \log \pi_{c_i}$ with $\lim\limits_{x \to 0+} x \log x = 0$ and $\pi_c = p(y = c|t)$

$P(y = W) = \frac{4}{10}$

$P(y = L) = \frac{6}{10}$

a)

$$\begin{aligned} i_H(y) &= -(p(y = W) \cdot \log p(y = W) + p(y = L) \cdot \log p(y = L)) \\ &= -(\frac{4}{10} \cdot \log \frac{4}{10} + \frac{4}{10} \cdot \log \frac{6}{10}) \\ &\approx 0.673012 \end{aligned}$$

Programming Task

Problem 5: See below

exercise_02_notebook

October 25, 2019

1 Programming assignment 1: k-Nearest Neighbors classification

[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).

1.3 Exporting the results to PDF

Once you complete the assignments, export the entire notebook as PDF and attach it to your homework solutions. The best way of doing that is 1. Run all the cells of the notebook. 2. Download the notebook in HTML (click File > Download as > .html) 3. Convert the HTML to PDF using e.g. https://www.sejda.com/html-to-pdf or `wkhtmltopdf` for Linux (tutorial) 4. Concatenate your solutions for other tasks with the output of Step 3. 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.

This way is preferred to using `nbconvert`, since `nbconvert` clips lines that exceed page width and makes your code harder to grade.

1.4 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`.

[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_state=123, test_size=(1 - split))
    return X_train, X_test, y_train, y_test
```

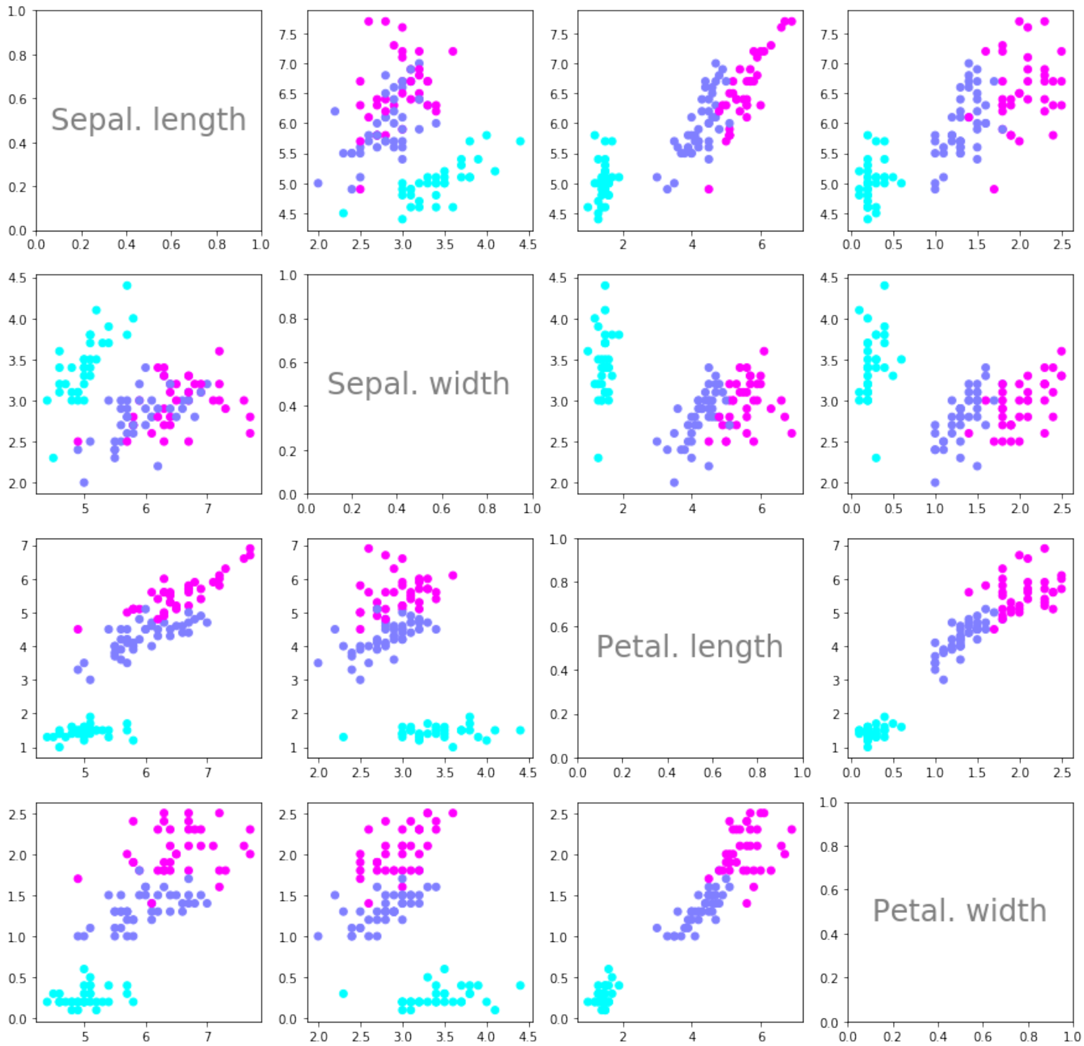
[3]:
```
# prepare data
split = 0.75
X_train, X_test, y_train, y_test = load_dataset(split)
```

1.5 Plot dataset

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

[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, alpha=.5)
        elif j == 1 and i == 1:
            axes[i,j].text(0.5, 0.5, 'Sepal. width', ha='center', va='center',␣
↪size=24, alpha=.5)
```

```
        elif j == 2 and i == 2:
            axes[i,j].text(0.5, 0.5, 'Petal. length', ha='center', va='center',␣
↪size=24, alpha=.5)
        elif j == 3 and i == 3:
            axes[i,j].text(0.5, 0.5, 'Petal. width', ha='center', va='center',␣
↪size=24, alpha=.5)
        else:
            axes[i,j].scatter(X_train[:,j],X_train[:,i], c=y_train, cmap=plt.cm.
↪cool)
```



1.6 Task 1: Euclidean distance

Compute Euclidean distance between two data points.

[5]:

```python
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.
    """
    result = x1 - x2
    result = result ** 2
    result = result.sum()
    result = result ** 0.5
    return result

# Tests
x1 = np.array((0,0,0,0))
x2 = np.array((0,0,0,1))

print(euclidean_distance(x1, x2))

x1 = np.array((0,0,1,0))
x2 = np.array((0,0,0,1))

print(euclidean_distance(x1, x2))

x1 = np.array((1,2,3,4))
x2 = np.array((4,3,2,1))

print(euclidean_distance(x1, x2))

x1 = np.array((0,0,0,0))
x2 = np.array((0,0,0,0))

print(euclidean_distance(x1, x2))
```

```
1.0
1.4142135623730951
4.47213595499958
0.0
```

1.7 Task 2: get k nearest neighbors' labels

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

[6]:
```
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.
    """
    distances = [(euclidean_distance(X_train[i], x_new), y_train[i]) for i in␣
 ↪range(len(X_train))]
    distances.sort()
    return np.array(distances[0:k])[:,1]

# Tests
unit_qube = np.array([
    [0,0,0,0],
    [0,0,0,1],
    [0,0,1,0],
    [0,0,1,1],
    [0,1,0,0],
    [0,1,0,1],
    [0,1,1,0],
    [0,1,1,1],
    [1,0,0,0],
    [1,0,0,1],
    [1,0,1,0],
    [1,0,1,1],
    [1,1,0,0],
    [1,1,0,1],
    [1,1,1,0],
    [1,1,1,1],
])
labels = np.array((0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15))
```

```
x_new = (0,0,0,0.5)
print(get_neighbors_labels(unit_qube, labels, x_new, 2))
x_new = (0,0,0.5,0.5)
print(get_neighbors_labels(unit_qube, labels, x_new, 4))
x_new = (0,0,0.5,0.5)
print(get_neighbors_labels(unit_qube, labels, x_new, 3))
```

```
[0. 1.]
[0. 1. 2. 3.]
[0. 1. 2.]
```

1.8 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. In case of a tie, choose the "lowest" label (i.e. the order of tie resolutions is $0 > 1 > 2$).

[11]:
```
def get_response(neighbors_labels, 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.
    """
    result = {}
    for neighbors_label in neighbors_labels:
        if neighbors_label not in result:
            result[neighbors_label] = 0
        result[neighbors_label] += 1

    result = sorted(result.items(), key=lambda item: item[1], reverse=True)
    return result[0][0]

# Tests
labels = np.array([1,1,1,2,2,2,3,3,3])
print(get_response(labels))
```

```
1
```

1.9 Task 4: compute accuracy

Compute the accuracy of the generated predictions.

[12]:
```
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.
    """
    correct_classified = np.sum(y_pred == y_test)
    return correct_classified / len(y_pred)
```

[13]:
```
# 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
```

1.10 Testing

Should output an accuracy of 0.9473684210526315.

[14]:

```
# prepare data
split = 0.75
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
```

Appendix

We confirm that the submitted solution is original work and was written by us without further assistance. Appropriate credit has been given where reference has been made to the work of others.

Munich, October 25, 2019, Signature Marcel Bruckner (03674122)

Munich, October 25, 2019, Signature Julian Hohenadel (03673879)

Munich, October 25, 2019, Signature Kevin Bein (03707775)