Machine Learning Exercise Sheet 2

k-Nearest Neighbors and Decision Trees

Group_369

Fan XUE – fan98.xue@tum.de

Xing ZHOU – xing.zhou@tum.de

Jianzhe LIU – jianzhe.liu@tum.de

November 3, 2021

Problem 1

According to the question we have:

A(1, 1) B(2, 0.5) C(1, 2.5) D(3, 3.5) E(5.5, 3.5) F(5.5, 2.5)

a) According to L_1-Norm,

$$d_1(X, Y) = \sum_i |X_i - Y_i|$$

we have for example:

$$d_{AB} = |1 - 2| + |1 - 0.5| = 1.5$$

By using this formula we can compute every distance between each two points and fill the results in a table:

L_1-**Norm**	**A**	**B**	**C**	**D**	**E**	**F**	**N-Neighbor**	**Class**
A	0.0	1.5	1.5	4.5	7.0	6.0	B or C	1
B	1.5	0.0	3.0	4.0	6.5	5.5	A	1
C	1.5	3.0	0.0	3.0	5.5	4.5	A	1
D	4.5	4.0	3.0	0.0	2.5	3.5	E	2
E	7.0	6.5	5.5	2.5	0.0	1.0	F	2
F	6.0	5.5	4.5	3.5	1.0	0.0	E	2

We can see that with leave-one-out cross validation, all the points are correctly classified.

b) According to L_2-Norm,

$$d_2(X,Y) = \left(\sum_i (X_i - Y_i)^2\right)^{\frac{1}{2}}$$

we have for example:

$$d_{AB} = ((1-2) + (1-0.5))^{\frac{1}{2}} = 1.118 \approx 1.2$$

Same as before, we still can use this formula to compute every distance between each two points and fill the results in another table:

L_1-**Norm**	**A**	**B**	**C**	**D**	**E**	**F**	**N-Neighbor**	**Class**
A	0.00	1.12	1.50	3.20	5.15	4.74	B	1
B	1.12	0.00	2.24	3.16	4.61	4.03	A	1
C	1.50	2.24	0.00	2.24	4.61	4.50	A	1
D	3.20	3.16	2.24	0.00	2.50	2.69	C	1
E	5.15	4.61	4.61	2.50	0.00	1.00	F	2
F	4.74	4.03	4.50	5.69	1.00	0.00	E	2

We can see that when we use L_2-Norm to validate our data, an error occurs: Point D is classified as Class 1 instead of 2.

c) According to these two cases we can draw the conclusion that a point can be classified into different class if we use different distance measurement. In our case, point D is close to Class 2 in the view of L_1-Norm yet close to Class 1 in the view of L_2-Norm.

Problem 2

a) If we use unweighted K-NN and $K_{new} = N_A + N_B + N_C$, then for X_{new}, the result would definitely be **Class C**. Because K-NN algorithm will always classify the new data into its major neighbor's class. If we use K that concludes every data in this training set, then the majority of the classes are equal to the major neighbors.

b) If we use weighted K-NN, whether the problem would be solved remains uncertain. If some cases it may, but in our case, we are lacking further information about our data's feature, thus not being able to conduct further calculations.

Problem 3

a) After observing this training model we can find that each data in this model has many features, and those features has huge differences on ranges. So if this model performs bad on test set, the reason could be followings:

1. Those features in different ranges have different impact on the result. Solution should be: Using data Standardization:

$$x_i^{'} = \frac{x_i - \mu_i}{\sigma_i}$$

or Mahalanobis distance:

$$\sqrt{(\mathbf{u}-\mathbf{v})^T \Sigma^{-1} (\mathbf{u}-\mathbf{v})}$$

2. Considering so many data, the choice of K can also affect the final result. Solution to this point is simple, we should use various K on validation set and find a better K for this model.
3. Still, when there are so many data, it's possible that a shift occurs between the training set and the test set. For example, in the training set there are only few samples about vans yet in the test set the majority of the samples are vans. Solution to this would be, equally divide the samples into different set to make minimize the shift.
4. When there are too many features in a single sample, it would bring the curse of dimension, which will lead to the lack of samples. As a result, the distance between samples will be amplified, and this could cause bad performance on test set. We can only use another algorithm to try to minimize this problem but we can't solve it so far.

b) If we use decision tree, the above-mentioned problems would be:

1. Features in different ranges will no longer affect the result, because in each node of the decision tree, only one feature will be discussed, that is to say, the other feature will have no impact on this exact node.
2. I would say, the parameter K in K-NN algorithm is just like the way we divide the decision trees. In K-NN, we can use some methods to find a better K, here in DT, we can also find a better way to divide the nodes, by using Misclassification rate, entropy or Gini index.
3. The shift on both sets also exists here and the solution is almost the same.
4. As said above, the curse of dimension will always more or less affect the result.

Problem 4

Can be **proved** by:

$$\begin{aligned} d_1(x,y)^2 &= \left(\sum_i |X_i - Y_i|\right)^2 \\ &= \sum_i (X_i - Y_i)^2 + 2\sum_i \sum_{j,j\neq i} |X_i - Y_i||X_j - Y_j| \\ &\geq \sum_i (X_i - Y_i)^2 \\ &= d_2(x,y)^2 \end{aligned}$$

Since d_1 and d_2 are all > 0, we can draw the conclusion that:

$$d_1(x,y) \geq d_2(x,y)$$

Problem 5

Can be **disproved** by following example:

Assuming that we have 3 different points: $y(2,\ 2)$ $x_1(1,\ 0.5)$ $x_2(2,\ 0)$

In L_2-Norm we have:

$$d_2(y,x_1) = \sqrt{(2-1)^2 + (2-0.5)^2} = 1.803 \approx 1.8$$

$$d_2(y,x_2) = \sqrt{(2-2)^2 + (2-0)^2} = 2$$

Obviously we have: $d_2(y,x_1) < d_2(y,x_2)$.

Let's say we only have these 3 points, then x_1 is the nearest neighbor of y in L_2-Norm.

But in L_1-Norm we have:

$$d_1(y,x_1) = |2-1| + |2-0.5| = 2.5$$

$$d_1(y,x_2) = |2-2| + |2-0| = 2$$

Obviously we have: $d_1(y,x_1) > d_1(y,x_2)$.

That is to say, in L_1-Norm, x_1 isn't the nearest neighbor of y.

Problem 6

Yes. We can define two new features $y_1 = x_1 - x_2$ and $y_2 = x_2$. By doing this, we can split the dataset by judging the condition $y_1 \leqslant 0$. With only one split the dataset is split into two parts and each part only has one sort of class. It means, there exists a decision tree of depth 1 that classifies this dataset with 100% accuracy.

Problem 7

a)

$$\begin{aligned} i_H(y) &= -p(y=W)\log p(y=W) - p(y=L)\log p(y=L) \\ &= -\frac{4}{10}\log\frac{4}{10} - \frac{6}{10}\log\frac{6}{10} \\ &= 0.971 \end{aligned}$$

b) Splitting by $x_1 = T$

$$\begin{aligned} \Delta i_H &= i_H(y) - p(x_1=T)i_H(x_1=T) - p(x_1=I)i_H(x_1=I) \\ &= 0.971 - \frac{1}{2}(-\frac{2}{5}\log\frac{2}{5} - \frac{3}{5}\log\frac{3}{5}) - \frac{1}{2}(-\frac{2}{5}\log\frac{2}{5} - \frac{3}{5}\log\frac{3}{5}) \\ &= 0 \end{aligned}$$

Splitting by $x_2 = M$

$$\begin{aligned} \Delta i_H &= i_H(y) - p(x_2=M)i_H(x_1=M) - p(x_2=P)i_H(x_2=P) \\ &= 0.971 - \frac{4}{10}(-\frac{2}{4}\log\frac{2}{4} - \frac{2}{4}\log\frac{2}{4}) - \frac{6}{10}(-\frac{2}{6}\log\frac{2}{6} - \frac{4}{6}\log\frac{4}{6}) \\ &= 0.020 \end{aligned}$$

Splitting by $x_3 = S$

$$\begin{aligned} \Delta i_H &= i_H(y) - p(x_3=S)i_H(x_3=S) - p(x_3=C)i_H(x_3=C) \\ &= 0.971 - \frac{1}{2}(-\frac{3}{5}\log\frac{3}{5} - \frac{2}{5}\log\frac{2}{5}) - \frac{1}{2}(-\frac{1}{5}\log\frac{1}{5} - \frac{4}{5}\log\frac{4}{5}) \\ &= 0.125 \end{aligned}$$

According to the calculation the split judgement will be $x_3 = S$, since in this case, the Δi_H is the biggest. If $x_3 = S$, the instance will be classified as W. Otherwise it will be classified as L.

Problem 8

Let $i^2 = \frac{125}{i}$, we get $i = 5$. Figure 1 shows the 2-d space of the dataset.

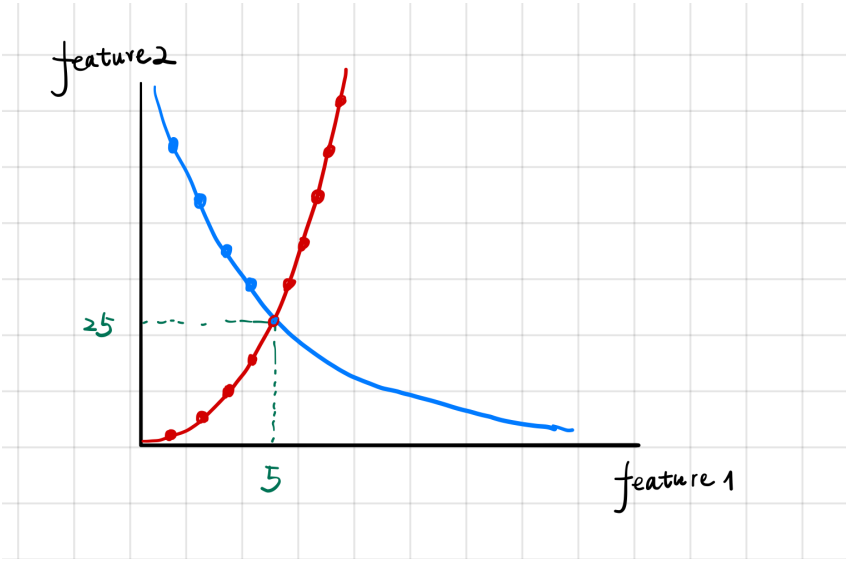


Figure 1: the 2-d space of the dataset

We can easily split the dateset into 4 parts. The first split uses the threshold feature2 $\leqslant$ 25.
The second split uses the threshold feature1 $\leqslant 5$ for both child nodes.
In this way, the depth of the decision tree is 2. Only one datapoint (5, 25) is missclassified, which is unavoidable.

Problem 9

The result of programming will be shown in next few pages.

exercise_02_notebook

November 3, 2021

1 Programming assignment 1: k-Nearest Neighbors classification

[ ]:
```
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.

You are only allowed to use the imported packages. Importing anything else is NOT allowed.

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).

In addition, we strongly recommend you to solve this task **without a single for loop**, i.e., only via vectorized (`numpy`) operations.

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. Export/download the notebook as PDF (File -> Download as -> PDF via LaTeX (.pdf)). 3. Concatenate your solutions for other tasks with the output of Step 2. 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.

Make sure you are using `nbconvert` Version 5.5 or later by running `jupyter nbconvert --version`. Older versions clip lines that exceed page width, which 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`.

[ ]:
```
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
```

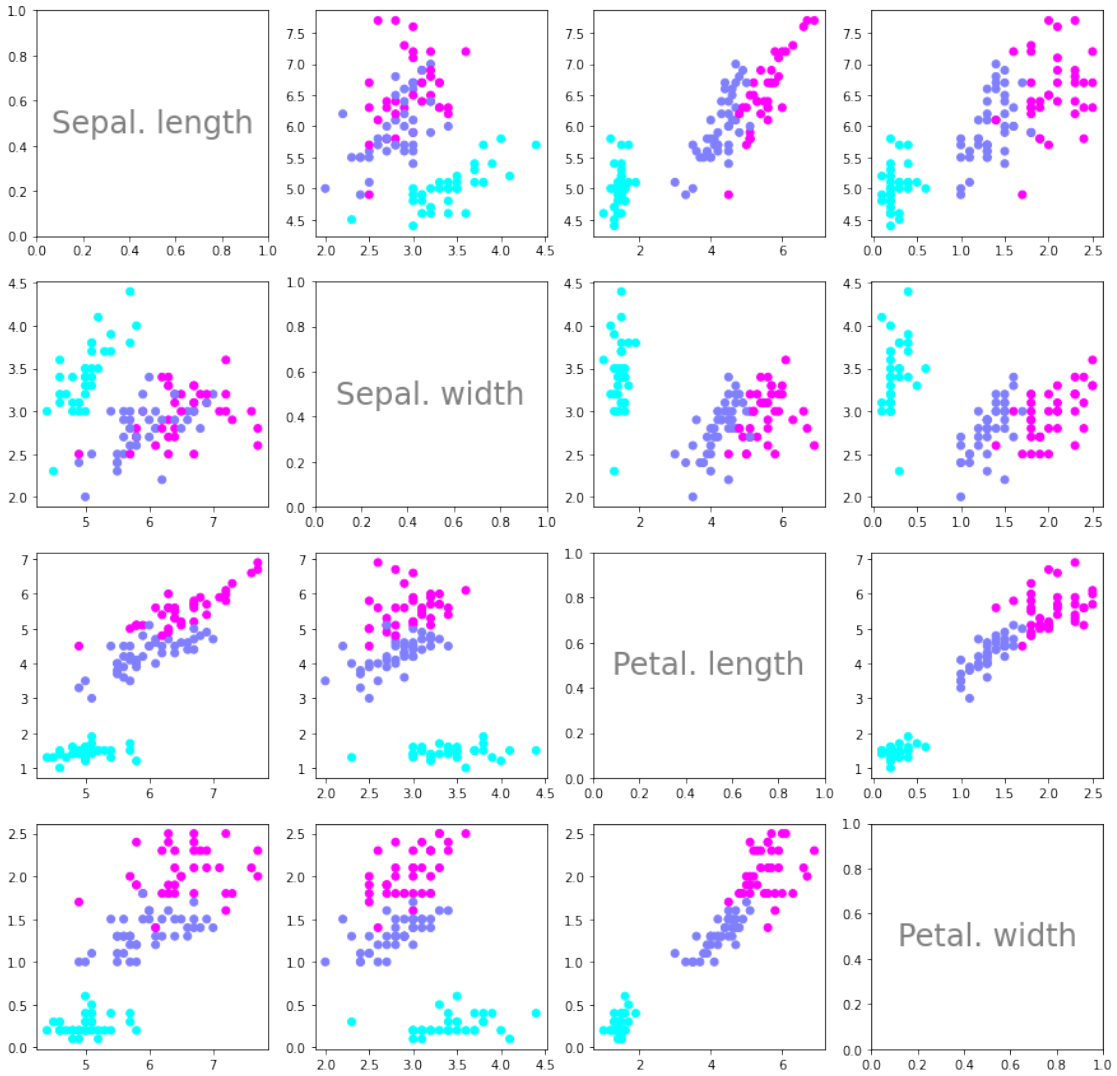
[ ]:
```
# prepare data
split = 0.75
X_train, X_test, y_train, y_test = load_dataset(split)
```

1.5 Plot dataset

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

[ ]:
```
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.

[ ]:
```python
def euclidean_distance(x1, x2):
    """Compute pairwise Euclidean distances between two data points.

    Parameters
    ----------
    x1 : array, shape (N, 4)
        First set of data points.
    x2 : array, shape (M, 4)
        Second set of data points.

    Returns
    -------
    distance : float array, shape (N, M)
        Pairwise Euclidean distances between x1 and x2.
    """
    # TODO
    n, m = x1.shape[0], x2.shape[0]
    dist = np.zeros([n, m])
    for i in range(n):
        for j in range(m):
            dist[i,j] = np.linalg.norm(x1[i,:] - x2[j,:])
    return dist
```

1.7 Task 2: get k nearest neighbors' labels

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

[ ]:
```python
def get_neighbors_labels(X_train, y_train, X_new, k):
    """Get the labels of the k nearest neighbors of the datapoints x_new.

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

    Returns
    -------
    neighbors_labels : array, shape (M, k)
```

```
        Array containing the labels of the k nearest neighbors.
    """
    # TODO
    dist = euclidean_distance(X_train, X_new)
    ind = np.argpartition(dist.T, k)[:,:k]
    neighbors_labels = y_train[ind]
    return neighbors_labels
```

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 $0 > 1 > 2$).

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

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

    Returns
    -------
    y : int array, shape (M,)
        Majority class among the neighbors.
    """
    # TODO
    #class_votes = np.zeros(num_classes)
    m = neighbors_labels.shape[0]
    y = np.zeros(m, np.int64)
    for row in range(m):
        counts = np.bincount(neighbors_labels[row,:])
        y[row] = np.argmax(counts)
    return y
```

1.9 Task 4: compute accuracy

Compute the accuracy of the generated predictions.

[ ]:
```
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.
    """
    # TODO
    correct = np.count_nonzero(y_pred == y_test)
    acc = correct/y_test.shape[0]
    return acc
```

[ ]:
```
# 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.
    """
    neighbors = get_neighbors_labels(X_train, y_train, X_test, k)
    y_pred = get_response(neighbors)
    return y_pred
```

1.10 Testing

Should output an accuracy of 0.9473684210526315.

[ ]:
```
# 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
```