NORMS AND DISTANCE REPORT

School or Department: Wu Han University

Grade and Specialty: Excellent engineer 2016

Name: Guo Yang, Kong Chuishun, Tang Rui

Advisor: Fangling Pu

April 1 2018

【**Abstract**】In this lab, we focus on the norm of a vector, a measure of its magnitude, and on related concepts like distance, angle and so on. The most important application of that is K-nearest-neighbors algorithm. So we will introduce it through a engineering problem by Python. The report consists of abstract, introduction, solution, result and conclusion.

【**Key words**】 Python; norm of a vector; K-nearest-neighbors algorithm

Table of Contents

Chapter 1 Introduction

1.1 Norm and distance

1.1.1 Norm

The Euclidean norm of an n-vector x (named after the Greek mathematician Euclid), denoted $||x||$, is the squareroot of the sum of the squares of its elements,

$$\|x\| = \sqrt{x1^2 + x2^2 + \cdots + xn^2} \quad 1.1.1$$

The Euclidean norm can also be expressed as the square root of the inner product of the vector with itself, i.e., $\|x\| = \sqrt{x^T x}$

Root-mean-square value. The norm is related to the root-mean-square (RMS) value of an n-vector x, defined as

$$\text{rms(x)} = \sqrt{\frac{\|x\|^2 + 2x^T x + \|y\|^2}{n}} = \frac{\|x\|^2}{\sqrt{n}} \quad 1.1.2$$

Norm of a sum: A useful formula for the norm of the sum of two vectors x and y is

$$\|x + y\| = \sqrt{\|x\|^2 + 2x^T x + \|y\|^2} \quad 1.1.3$$

1.1.2 Distance

Euclidean distance. We can use the norm to define the Euclidean distance between two vectors a and b as the norm of their difference:

$$\textbf{dist}(a, b) = \|a - b\|.$$

For one, two, and three dimensions, this distance is exactly the usual distance between points with coordinates a and b, as illustrated in figure 1.1. If a and b are n-vectors, we refer to the RMS value of the difference, $\frac{\|a-b\|}{\sqrt{n}}$, as the RMS deviation between the two vectors.

When the distance between two n-vectors x and y is small, we say they are ‘close’ or ‘nearby’, and when the distance $\|x - y\|$. is large, we say they are ‘far’. The particular numerical values of $\|x - y\|$ that correspond to ‘close’ or ‘far’ depend on the particular application.

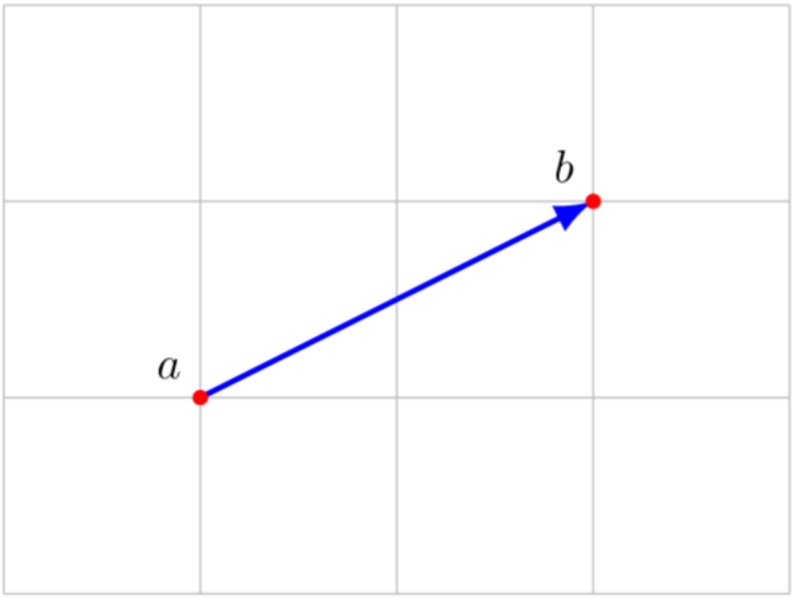

Figure 1.1 The norm of the displacement b−a is the distance between the points with coordinates a and b.

Triangle inequality. We can now explain where the triangle inequality gets its name. Consider a triangle in two or three dimensions, whose vertices have coordinates a, b, and c. The lengths of the sides are the distances between the vertices,

dist(a, b) = $\|a-b\|$, dist(b, c) = $\|b-c\|$, dist(a, c) = $\|a-c\|$

Geometric intuition tells us that the length of any side of a triangle cannot exceed the sum of the lengths of the other two sides. For example, we have

$$\|a-c\| \leqslant \|a-b\| + \|b-c\|. \qquad 1.1.4$$

This follows from the triangle inequality, since

$$\|a-c\| = \|(a-b)+(b-c)\| \leq \|(a-b))\| + \|(b-c))\|. \qquad 1.1.5$$

This is illustrated in figure 1.2.

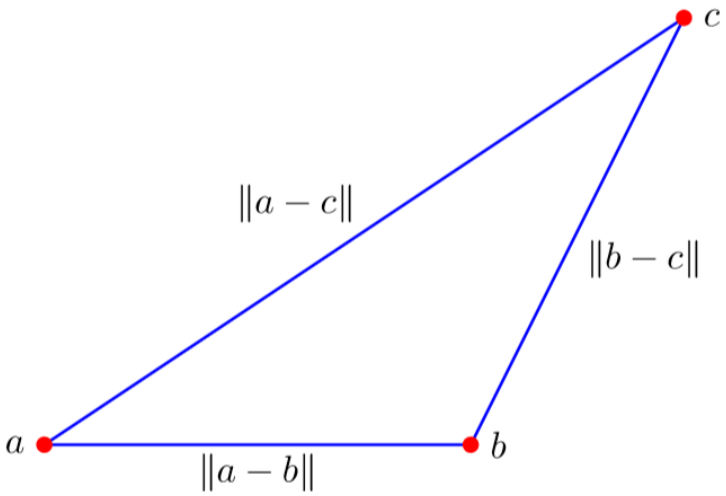

Figure 1.2 Triangle inequality

1.2 K-nearest-neighbors algorithm

In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression.[1] In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression:

- In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.
- In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

Condensed nearest neighbor (CNN, the Hart algorithm) is an algorithm designed to reduce the data set for k-NN classification.[20] It selects the set of prototypes U from the training data, such that 1NN with U can classify the examples almost as accurately as 1NN does with the whole data set.

Given a training set X, CNN works iteratively:

1. Scan all elements of X, looking for an element x whose nearest prototype from U has a different label than x.
2. Remove x from X and add it to U
3. Repeat the scan until no more prototypes are added to U.

Below is an illustration of CNN in a series of figures. There are three classes (red, green and blue).

Fig1.3: initially there are 60 points in each class.

Fig1.4 shows the 1NN classification map: each pixel is classified by 1NN using all the data.

Fig1.5 shows the 5NN classification map. White areas correspond to the unclassified regions, where 5NN voting is tied (for example, if there are two green, two red and one blue points among 5 nearest neighbors).

Fig1.6 shows the reduced data set. The crosses are the class-outliers selected by the (3,2) NN rule (all the three nearest neighbors of these instances belong to other classes); the squares are the prototypes, and the empty circles are the absorbed points. The left bottom corner shows the numbers of the class-outliers, prototypes and absorbed points for all three classes. The number of prototypes varies from 15% to 20% for different classes in this example.

Fig1.7 shows that the 1NN classification map with the prototypes is very similar to that with the initial data set. The figures were produced using the Mirkes applet.

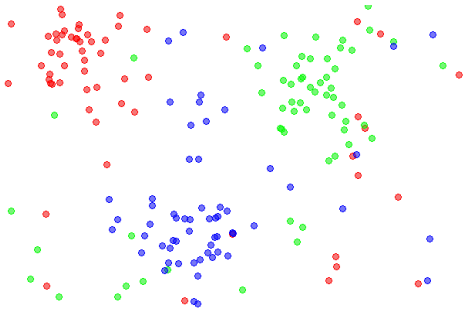

Fig. 1.3. The dataset.

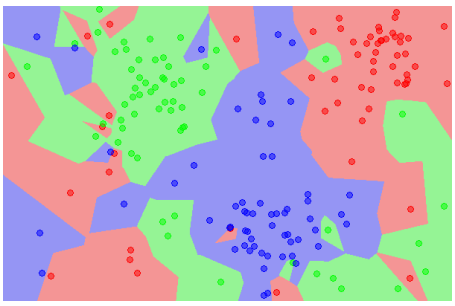

Fig. 1.4. The 1NN classification map.

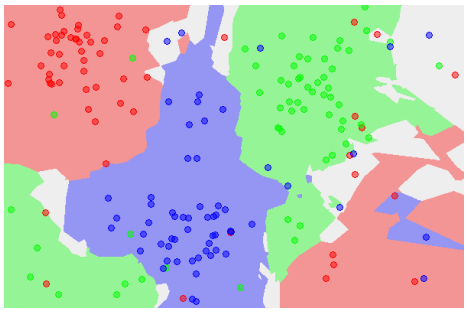

Fig. 1.5. The 5NN classification map.

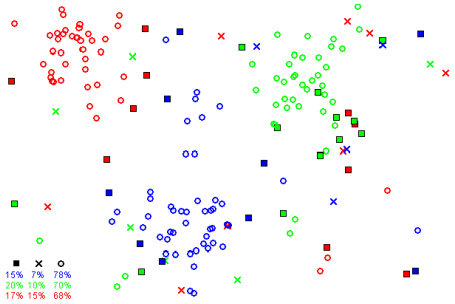

Fig. 1.6 The CNN reduced dataset.

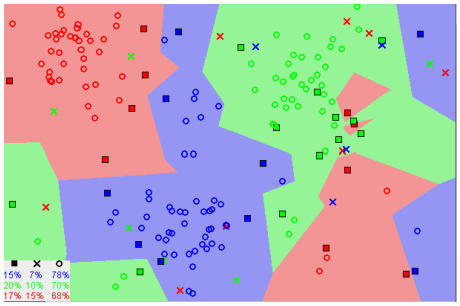

Fig. 1.7. The 1NN classification map based on the CNN extracted prototypes.

Chapter 2 Introduction of the lab

2.1 Exploring KNN in Code

The data set we'll be using is the Iris Flower Dataset (IFD) which was first introduced in 1936 by the famous statistician Ronald Fisher and consists of 50 observations from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features

were measured from each sample: the length and the width of the sepals and petals. Our goal is to train the KNN algorithm to be able to distinguish the species from one another given the measurements of the 4 features.

```
# ============================== data preprocessing ==================================
# define column names
names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']

# loading training data
df = pd.read_csv('C:\Users\LENOVO-PC\Desktop\iris.data.txt', header=None, names=names)
print(df.head())
```

Next, We'll use scikit-learn to train a KNN classifier and evaluate its performance on the data set using the 4 step modeling pattern:

1. Import the learning algorithm
2. Instantiate the model
3. Learn the model
4. Predict the response

```
# create design matrix X and target vector y
X = np.array(df.ix[:, 0:4])     # end index is exclusive
y = np.array(df['class'])   # showing you two ways of indexing a pandas df

# split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
# ============================== KNN with k = 3 ============================================
# instantiate learning model (k = 3)
knn = KNeighborsClassifier(n_neighbors=3)
```

Finally, following the above modeling pattern, we define our classifier, in this case KNN, fit it to our training data and evaluate its accuracy. We'll be using an arbitrary K but we will see later on how cross validation can be used to find its optimal value.

```
# split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state
# ============================== KNN with k = 3 ====================================
# instantiate learning model (k = 3)
knn = KNeighborsClassifier(n_neighbors=3)

# fitting the model
knn.fit(X_train, y_train)

# predict the response
pred = knn.predict(X_test)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the knn classifier for k = 3 is %d%%' % acc)
```

2.2 Parameter Tuning with Cross Validation

In this section, we explore a method that can be used to *tune* the hyperparameter K.

Obviously, the best K is the one that corresponds to the lowest test error rate, so let's suppose we carry out repeated measurements of the test error for different values of K. Inadvertently, what we are doing is using the test set as a training set! This means that we are underestimating the true error rate since our model has been forced to fit the

test set in the best possible manner. Our model is then incapable of generalizing to newer observations, a process known as **overfitting**. Hence, touching the test set is out of the question and must only be done at the very end of our pipeline.

If that is a bit overwhelming for you, don't worry about it. We're gonna make it clearer by performing a 10-fold cross validation on our dataset using a generated list of odd K's ranging from 1 to 50.

```
# creating odd list of K for KNN
myList = list(range(0,50))
neighbors = list(filter(lambda x: x % 2 != 0, myList))

# empty list that will hold cv scores
cv_scores = []

# perform 10-fold cross validation
for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=10, scoring='accuracy')
    cv_scores.append(scores.mean())
```

We specify that we are performing 10 folds with the cv=10 parameter and that our scoring metric should be accuracy since we are in a classification setting.

Finally, we plot the misclassification error versus K.

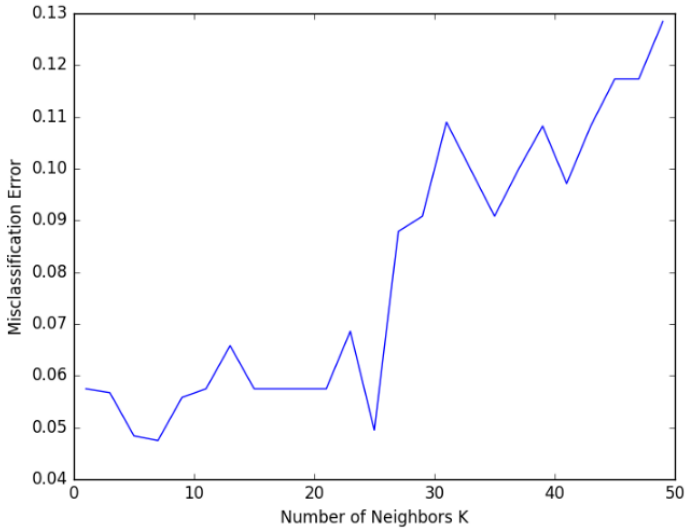

Figure 2.2.1

Figure 2.2.1 tells us that K=7 results in the lowest validation error.

Chapter 3 Results and Conclusion

The result is shown in the Figure 3.1

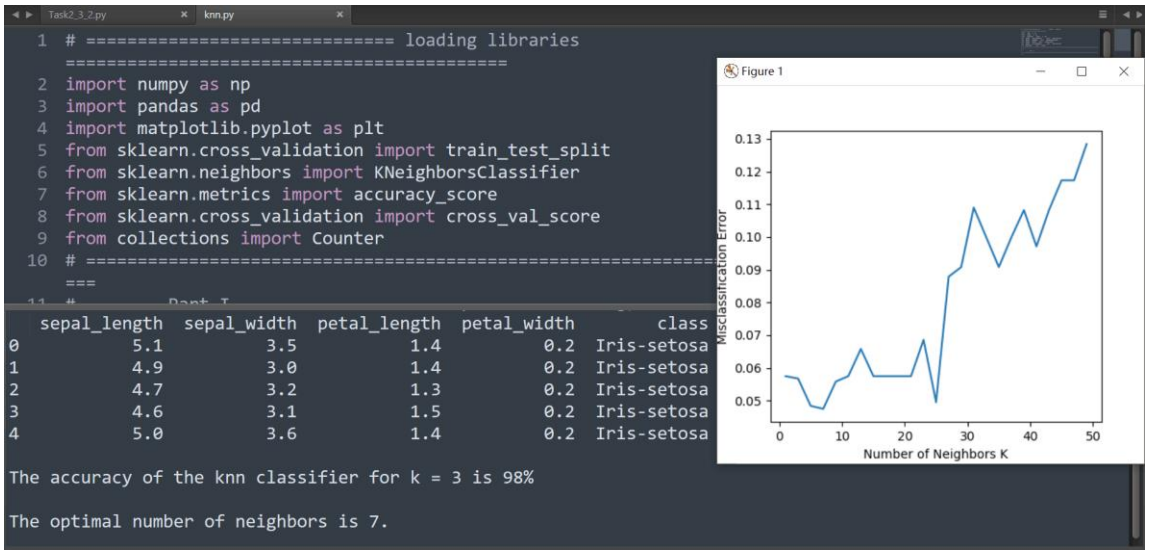

Figure 3.1

We can get that:

When K=7, the validation error is lowest.

When K=3, the accuracy is 98%.

Bibliography

[1] A Complete Guide to K-Nearest-Neighbors with Applications in Python and R

Kevin Zakka

[2] https://en.wikipedia.org

Appendix

The python code:

#==========loading libraries==========

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cross_validation import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score

from sklearn.cross_validation import cross_val_score

from collections import Counter

```
#
================================================================
================================
#                       Part I
#
================================================================
================================

#          ========================          data          preprocessing
==========================================

# define column names
names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']

# loading training data
df = pd.read_csv('iris.data.txt', header=None, names=names)
print(df.head())

# create design matrix X and target vector y
X = np.array(df.ix[:, 0:4])       # end index is exclusive
y = np.array(df['class'])    # showing you two ways of indexing a pandas df

# split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
#       ============================          KNN          with          k          =          3
================================================

# instantiate learning model (k = 3)
knn = KNeighborsClassifier(n_neighbors=3)

# fitting the model
knn.fit(X_train, y_train)
```

```
# predict the response
pred = knn.predict(X_test)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the knn classifier for k = 3 is %d%%' % acc)
# ================================ parameter tuning ==========================================

# creating odd list of K for KNN
myList = list(range(0,50))
neighbors = list(filter(lambda x: x % 2 != 0, myList))

# empty list that will hold cv scores
cv_scores = []

# perform 10-fold cross validation
for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=10, scoring='accuracy')
    cv_scores.append(scores.mean())

# changing to misclassification error
MSE = [1 - x for x in cv_scores]

# determining best k
optimal_k = neighbors[MSE.index(min(MSE))]
print('\nThe optimal number of neighbors is %d.' % optimal_k)

# plot misclassification error vs k
```

```
plt.plot(neighbors, MSE)
plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()
#
=================================================================
==================================
#                          Part II
#
=================================================================
==================================
#  ==================================== writing our own KNN
==================================
'''
def train(X_train, y_train):
    # do nothing
    return

def predict(X_train, y_train, x_test, k):
    # create list for distances and targets
    distances = []
    targets = []

    for i in range(len(X_train)):
        # first we compute the euclidean distance
        distance = np.sqrt(np.sum(np.square(x_test - X_train[i, :])))
        # add it to list of distances
        distances.append([distance, i])

    # sort the list
```

```
    distances = sorted(distances)

    # make a list of the k neighbors' targets
    for i in range(k):
        index = distances[i][1]
        #print(y_train[index])
        targets.append(y_train[index])

    # return most common target
    return Counter(targets).most_common(1)[0][0]

def kNearestNeighbor(X_train, y_train, X_test, predictions, k):
    # check if k is not larger than n
    if k > len(X_train):
        raise ValueError

    # train on the input data
    train(X_train, y_train)

    # predict for each testing observation
    for i in range(len(X_test)):
        predictions.append(predict(X_train, y_train, X_test[i, :], k))
# ================================ testing our KNN
================================================

# making our predictions
predictions = []
try:
    kNearestNeighbor(X_train, y_train, X_test, predictions, 7)
    predictions = np.asarray(predictions)
```

```
    # evaluating accuracy
    accuracy = accuracy_score(y_test, predictions) * 100
    print('\nThe accuracy of OUR classifier is %d%%' % accuracy)

except ValueError:
    print('Can\'t have more neighbors than training samples!!')
'''
```