# Makine Öğrenmesi - Homework 2

Mahmut Bulut - 14501026
Computer Engineering Dept., Yıldız Technical University

December 21, 2014

## Yöntem

a) **k-NN** ile yapılan sınıflandırma işlemi her test seti için train edilen diğer kfold setlerinden(her iterasyon için 9 adet) uygulanması ve bu setlerin *knn* fonksiyonuna girdi olarak girmesinden önce k-fold ile ayrılması esasına dayanmıştır. k-NN fonksiyonuna setler verilmeden önce ayrım yapılmalı ve setler sonra verilmelidir.

Kodda bulunan k-NN yapılanmasının kaba kod'u aşağıdaki gibidir.

**Algorithm 1** k-NN algorithm

```
1: procedure KNN(kvalue, trainset, testset, classdata)   ▷ k değeri, setler ve doğru sınıfları ver
2:     predicted_class ← []
3:     for each test in testset do          ▷ Her test setindeki değer için trainset ile training yap
4:         distances ← []
5:         for each train in trainset do
6:             distances += euclidean_distance(test, train)   ▷ Öklid uzaklıklarını uzaklıklara ekle
7:         end for
8:         k_nn ← sorted(distances)                  ▷ Uzaklıkları küçükten büyüğe sırala
9:         predicted_class+=classify(k_nn, classdata)        ▷ Yakındaki set verilerini sınıfla
10:    end for
11:    return predicted_class                                ▷ Tahmin edilen sınıfı döndür
12: end procedure
```

b) **k-fold** ile cross validation 3823 satırlık verinin fold sayısına göre floor fonksiyonu ile aşağı doğru yuvarlanarak setteki eleman sayısının hesaplanması ve her setin parçalarının iterate edilmesi üzerinden çalışmaktadır. Set'in parçaları fold sayısına göre esneklik gösterecek şekilde hesaplanılmaktadır. Hesaplama kodu aşağıda verilmiştir. Ayrıca kfold kodu 65. indeksteki gerçek sınıf değerlerini çıkartmakta ve onları ayrı listelerde bize karşılaştırmamız için vermektedir bu bize hız kazandırmaktadır.

```
testslicecount = int(math.floor(allresults.__len__()/fold_count))
```

Kodda bulunan k-NN yapılanmasının kaba kod'u aşağıdaki gibidir.

**Algorithm 2** k-fold algorithm

**procedure** KFOLD($allresults, fold\_count$) ▷ Veriyi ve fold sayısını ver
  $slices \leftarrow []$
  $index \leftarrow 0$
  $slicecount \leftarrow floor(length(results)/fold\_count)$ ▷ Her folddaki element sayısı
  $training\_set, testing\_set, expects\_train, expects\_test \leftarrow []$ ▷ Listeleri initialize et
  **for** $x = 0$, $x$++, while $x < fold\_count$ **do**
    **append** $partition(allresults, [ind, ind + slicecount])$ **to** $slices$
    $ind$+=$slicecount$
    **extend** $slices$ **with** other slice partitions except testing
    **append** true(expected) classes of training partition at index 65 **to** $expects\_train$ ▷ Doğru sınıflar train için
    **append** true(expected) classes of testing partition at index 65 **to** $expects\_text$ ▷ Doğru sınıflar testing için
    **append** training slice **to** $training\_set$
    **append** testing slice **to** $testing\_set$
  **end for**
  **return** $training\_set, testing\_set, expects\_train, expects\_test$
**end procedure**

## Uygulama

a) 10-fold sistemi kullanılarak iterasyonlar yapıldı ve yaklaşık olarak aynı başarımda koşum ortaya çıktı. İterasyon çıktıları aşağıdaki gibidir. 10 iterasyonun başarımları listede gösterilmiş ve meanleri alınmıştır. Başarım k sayısı arttıkça artması gerekirken çok küçük oranda düşüş görülmüştür.

```
PERFORMANCE for K=1:
[0.9738219895287958, 0.9947643979057592, 0.9869109947643979, 0.9842931937172775,
0.9764397905759162, 0.9869109947643979, 0.9842931937172775, 0.9921465968586387,
 0.9869109947643979, 0.981675392670157]
PERFORMANCE AVERAGE: 0.984817


PERFORMANCE for K=3:
[0.9581151832460733, 0.9921465968586387, 0.9869109947643979, 0.981675392670157,
0.9738219895287958, 0.9947643979057592, 0.9895287958115183, 0.9921465968586387,
0.9869109947643979, 0.9764397905759162]
PERFORMANCE AVERAGE: 0.983246


PERFORMANCE for K=5:
[0.9685863874345549, 0.9895287958115183, 0.9842931937172775, 0.9842931937172775,
0.9764397905759162, 0.9921465968586387, 0.9842931937172775, 0.9895287958115183,
0.981675392670157, 0.9764397905759162]
```

```
PERFORMANCE AVERAGE: 0.982723
```

En iyi k değeri K=1 ile yüksek başarım göstermiştir. Aslında K=5 iken en iyi başarım göstermesi gerekmektedir ama set'in değerleri dağıtık olduğu için 1 en iyi değer olarak gözükmektedir.

b) Diagonal sayıların büyüklüğü doğru sınıflandırıldığını göstermektedir ayrıca diagonal sayıların k arttıkça azalması ve satırlardaki diğer kısımlara dağılması ise k değerinin arttıkça yanlış sınıflandırmanın da arttığını bize göstermektedir.

```
[374, 0, 0, 0, 1, 0, 1, 0, 0, 0]
[0, 384, 1, 1, 0, 0, 1, 1, 0, 1]
[0, 0, 379, 0, 0, 0, 0, 0, 1, 0]
[0, 2, 1, 382, 0, 1, 0, 0, 1, 2]
[0, 0, 0, 0, 383, 0, 1, 1, 0, 2]
[0, 1, 0, 1, 0, 369, 0, 0, 1, 4]
[0, 2, 0, 0, 0, 0, 373, 0, 0, 0]
[0, 0, 0, 1, 1, 0, 0, 384, 0, 0]
[1, 10, 1, 2, 1, 0, 0, 0, 364, 1]
[0, 3, 0, 1, 1, 2, 0, 4, 1, 370]
375 7
PERFORMANCE for K=1:
[0.9738219895287958, 0.9947643979057592,
0.9869109947643979, 0.9842931937172775, 0.9764397905759162, 0.9869109947643979,
0.9842931937172775, 0.9921465968586387, 0.9869109947643979, 0.981675392670157]
PERFORMANCE AVERAGE: 0.984817

[374, 0, 0, 0, 1, 0, 1, 0, 0, 0]
[0, 384, 2, 0, 0, 0, 0, 1, 0, 2]
[0, 0, 378, 1, 0, 0, 0, 0, 1, 0]
[0, 1, 0, 382, 0, 1, 0, 0, 0, 5]
[0, 0, 0, 0, 380, 0, 2, 1, 0, 4]
[0, 0, 0, 2, 0, 367, 0, 0, 0, 7]
[0, 2, 0, 0, 0, 0, 373, 0, 0, 0]
[0, 2, 0, 1, 1, 0, 0, 382, 0, 0]
[0, 11, 0, 1, 1, 0, 1, 0, 365, 1]
[0, 3, 0, 3, 1, 1, 0, 2, 1, 371]
373 9
PERFORMANCE for K=3:
[0.9581151832460733, 0.9921465968586387,
 0.9869109947643979, 0.981675392670157, 0.9738219895287958, 0.9947643979057592,
 0.9895287958115183, 0.9921465968586387, 0.9869109947643979, 0.9764397905759162]
PERFORMANCE AVERAGE: 0.983246

[374, 0, 0, 0, 1, 0, 1, 0, 0, 0]
[0, 384, 2, 0, 0, 0, 0, 1, 0, 2]
```

```
[0, 1, 379, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 381, 0, 1, 0, 1, 1, 4]
[1, 0, 0, 0, 379, 0, 2, 1, 0, 4]
[0, 0, 0, 2, 0, 367, 0, 0, 0, 7]
[0, 2, 0, 0, 0, 0, 373, 0, 0, 0]
[0, 2, 0, 1, 1, 0, 0, 382, 0, 0]
[0, 12, 0, 1, 0, 1, 0, 0, 365, 1]
[0, 3, 0, 3, 2, 1, 0, 2, 1, 370]
373 9
PERFORMANCE for K=5:
[0.9685863874345549, 0.9895287958115183, 0.9842931937172775, 0.9842931937172775,
0.9764397905759162, 0.9921465968586387, 0.9842931937172775, 0.9895287958115183,
0.981675392670157, 0.9764397905759162]
PERFORMANCE AVERAGE: 0.982723
```

Örnek olarak 4 rakamını ele alalım. Başta 383, 380 ve 379 olarak azalarak doğru tanınmıştır. Diğer tanınmalar artıp azalan doğru tanınma oranları sergilemektedir. Soldan sağ alta diagonal bağlantının yüksek olması algoritmanın verimli çalıştığını göstermektedir. Overall'da başarım eşik değerlerinin üzerinde gözükmektedir.

## Sonuç

a) **k-NN** ile sayı tanıma işlemi verimlidir ortalama %98 gibi bir başarım (k = [1, 3 ve 5] serisinde) sayı tanıma işleminin knn algoritması yardımıyla doğru yapılabildiğini göstermektedir. İnternetteki araştırmalarım sonucunda *Euclidean*'ın yanında *Minkowski* ve *Mahalanobis*[1] uzaklığının da güzel sonuçlar verebildiğini görmüş oldum. *Minkowski* uzaklığını polynomial 1 ile denedim. Euclidean kadar başarılı olmasa da k'dan dolayı oluşan verim farklarını ortaya çıkarttı.

b) k-NN ve k-fold'un birlikte kullanıldığı kodlar aşağıdaki gibidir. Kod python kodudur ve versiyon 2.7 ile denenmiştir. Ayrıca Hy(python lisp dialect) dilinde yazmaya başladığım kod da GitHub'daki depomda[2] bulunmaktadır.

```
__author__ = 'vertexclique'

# Run with below command to inspect the output properly
# time python -u mlhwtwo.py | tee mlhwtwooutput.txt

import math
import numpy as np
from scipy.spatial import distance

def readcsv():
```

[1] http://www.csee.umbc.edu/~tinoosh/cmpe650/slides/K_Nearest_Neighbor_Algorithm.pdf
[2] http://github.com/vertexclique

```
    """
    Read CSV. Doesn't need to user csv lib.
    :return: results in row-based list
    """
    allresults = []
    with open('sayi.dat') as f:
        for line in f:
            text = line.rstrip("\n").split(",")
            result = map(int, text)
            allresults.append(result)
        return allresults

def euclidean_distance(sample1, sample2):
    """
    Use euclidean distance to calculate set distances
    Also minkowski gives good results with polynomial 1

    :param sample1: First sample
    :param sample2: Second sample
    :return: distance between them
    """
    return distance.euclidean(sample1, sample2)

def evaluate(calculated, expected, matrix_10x10):
    """
    Fill the fold matrix and evaluate the predicted class

    :param calculated: k-NN calculated class
    :param expected: expected class at index 65
    :param matrix_10x10: 10x10 matrix for inspection of all
    :return: evaluated results and matrix itself
    """
    result = []

    # If expected length is not equal to calculated one dismiss it
    if calculated.__len__() != expected.__len__():
        raise "Error on calculation"
    else:
        for x in range(0, calculated.__len__()):
            # Substract two data to find which one is correct as class
            result.append(calculated[x] - expected[x])
            # Calculate 10x10 matrix for folding
            matrix_10x10[expected[x]][calculated[x]] += 1

    # make array of evaluation results
```

```
    eval_result = np.zeros(2,int)
    for x in result:
        # increment the correct prediction by 1
        if x == 0:
            eval_result[0] += 1
        # increment the wrong prediction by 1
        else:
            eval_result[1] += 1
    # return evaluation result
    return eval_result, matrix_10x10

def knn(kvalue, trainset, testset, classdata):
    """
    k-NN calculation

    :param kvalue: k-NN kvalue
    :param trainset: training set
    :param testset: testing set
    :param classdata: true class data
    :return: calculated classes of testing data
    """
    pred_class = []
    for testi, testd in enumerate(testset):
        distances = []
        for traini, traind in enumerate(trainset):
            # append euclidean distances to be sorted
            distances.append((euclidean_distance(testd, traind), traini))

        # sort distances inversely from smallest to largest and take kvalue times.
        k_nn = sorted(distances)[:kvalue]

        pred_class.append(classify(k_nn, classdata))

    # return prediction class
    return pred_class

def most_common(lst):
    """
    Find most common element in list so we can classify in_circle elements
    :param lst: list to find the most common element in it
    :return: most common element...
    """
    return max(set(lst), key=lst.count)

def classify(selected_knn, class_data):
```

```
    """
    Classify selected instances into class of them
    We can call them in_circle

    :param selected_knn: reached number of k elements
    :param class_data: class data of elements
    :return: the most common element in that class so sleected k-nn will be that
    """

    in_circle = []
    for index, value in selected_knn:
        in_circle.append(class_data[value])

    return most_common(in_circle)

def partition(waiting_list, indices):
    """
    Split a list into partition with given indexes

    :param waiting_list: list to be partitioned based on indices
    :param indices: indices that slices into pieces
    :return: slices...
    """
    return [waiting_list[i:j] for i, j in zip([0]+indices, indices+[None])]

def extract_classes(general_row):
    """
    Extract true classes from the end of data
    and make a classes list from them

    :param general_row: just a 65 indexed row
    :return: copy the last one and return it (it is not `.pop` method nor `del`)
    """
    classes = []
    for ex in general_row:
        classes.append(ex[-1])

    return classes

def kfold_partitioner(allresults, fold_count):
    """
    k-FOLD code to make folding properly
    It is flexible as you can see, it takes fold count
    and folds in for that count
```

```
    :param allresults: all data used in experimenting
    :param fold_count: fold count for slices
    :return: slices based on list of lists
    """
    slices = []
    ind = 0

    testslicecount = int(math.floor(allresults.__len__()/fold_count))

    training_set, testing_set, expects_train, expects_test = ([] for i in range(4))

    for x in xrange(0, fold_count):
        slices.append(partition(allresults, [ind, ind+testslicecount]))
        ind += testslicecount
        # print(slices[x][1].__len__()) # Going to be test data
        # slices [x][1] is going to be test data
        # Extend the list of first slice with others rather than testing set
        slices[x][0].extend(slices[x][2])
        # print(slices[x][0].__len__()) # Going to be train data
        # slices [x][0] is going to be train data
        expects_train.append(extract_classes(slices[x][0]))
        expects_test.append(extract_classes(slices[x][1]))

        training_set.append(slices[x][0])
        testing_set.append(slices[x][1])

    return training_set, testing_set, expects_train, expects_test


if __name__ == "__main__":
    # Read it first
    allresults = readcsv()

    # k-fold number
    foldnumber = 10

    # k-values for testing
    kvalues = [1, 3, 5]

    # Did i remove trailing true classes?
    remove_trailing = False
    # Store final evaluation results
    final_results = []
    # Store performance results
    performance = []
```

```
# Apply k-fold partitioning to data set
train, test, expected_train, expected_test = kfold_partitioner(allresults=allresults,

for kval in kvalues:
    matrix_10x10 = [[0 for i in range(10)] for _ in range(10)]
    # Cross validate with k-fold
    for x in xrange(0, foldnumber):
        # Remove trailing expected class values from sets at first iteration
        if remove_trailing == False:
            for row in train[x]:
                del row[-1]
            for row in test[x]:
                del row[-1]
            remove_trailing = True

        # Run k-NN and evaluate results
        pred_class = knn(kval, train[x], test[x], expected_train[x])
        eval_result, res_matrix_10x10 = evaluate(pred_class, expected_test[x], matrix

        # Print result matrix and inspect it
        for a in res_matrix_10x10:
            print a

        # Print prediction class set and expected set just for inspection
        # print "==="
        # print(pred_class)
        # print(expected_test[x])
        # print "==="

        # Print how many is tru how many is false in evaluation and
        # calculate performance from evaluation results
        final_results.append(eval_result[0])
        final_results.append(eval_result[1])
        print final_results[0], final_results[1]
        performance.append(float(final_results[0]) / float(final_results[0] + final_r
        final_results = []
        print "PERFORMANCE for K=%i: " % kval
        print(performance)
    print "PERFORMANCE AVERAGE: %f" % np.mean(performance)
    performance = []
```