Notes on understanding classifiers

[work in progress]

Tomasz Golan

January 27, 2016

Abstract

This note is related to Simple Classifier project (https://github.com/TomaszGolan/simpleClassifiers) which was an exercise to understand two classifiers: k-Nearest Neighbors (kNN) and Support Vector Machine (SVM) . Two classes of 2D points are considered within the project: *separable* (below or above $f(x) = x$)) and *inseparable* (inside or outside circle). The purpose of this note is to explain technical details of both algorithms and demonstrate how they work on simple examples. Please note, I am not an expert in the field and the note is rather how I understand the problem.

1 Introduction

A classifier, in machine learning, is an algorithm for finding a class of studying object, based on a set of objects with known class (training set). Thus, it is a type of supervised learning.

If a classification can only distinguish between two training sets it is called binary classification, otherwise it is multiclass classification. SVM is a binary classifier, so several SVMs combination is required for multiclass classification. KNN is a multiclass classifier. In this note, only binary problems are considered, so combining binary classifiers problem is not discussed.

For any supervised learning the choice of a training set is crucial. They should uniformly cover the whole phase space. The optimal size of the training set is unique for a problem. It depends on the classification method, the complexity of the problem, and on the separability of classes. To determine the optimal size of the training set one can use learning curves, which presents the error of classification as a function of the training set size.

For complex problems, the choice of the proper features to feed a classifier is important and non-trivial. They must be chosen to well define a membership to some class. Unless you are a genius you will probably need to make some tests to determine the right set of properties to learn a classifier. In this note, only simple examples are discussed, so there is no problem with the choice of features.

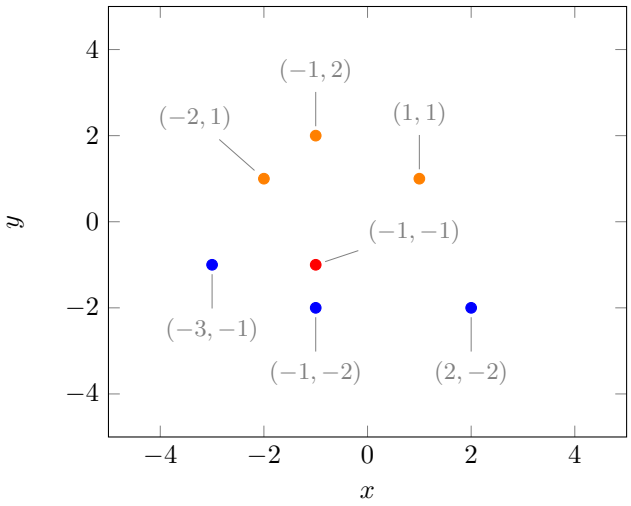


Figure 1: Two classes of points: blue and orange. The red dot represents the point to classify.

class	coordinates	distance^2
blue	$(-3, -1)$	4
blue	$(-1, -2)$	1
blue	$(2, -2)$	10
orange	$(-1, 2)$	9
orange	$(1, 1)$	8
orange	$(-2, 1)$	9

Table 1: The list of distances between the red point and each training example from Fig. 1.

2 k-Nearest Neighbors

k-Nearest Neighbors is one of the simplest machine learning algorithms[1]. Lets assume N training examples in our feature space are given by:

- a feature vector $\vec{x} = (x_1, x_2, ..., x_n)$
- a class membership y

For a given vector $\vec{x}'$ its membership y' must be determined. The idea is to look at k nearest neighbors and let the majority decide - if among the k nearest neighbors the most of training vectors belongs to the class y_j, the given vector also belongs to y_j. The optimal value of k is unique for the problem and must be determine empirically.

Lets consider the example presented on Fig. 1. There are two classes of points: blue and orange. The size of the training set is 6 (3 for each class). The red point is the one to classify (by eye it looks like it should belongs to blue points).

[1] kNN is also used for regression. In this note, only the use for classification is discussed.

class	coordinates	distance^2	weight
blue	$(-3, -1)$	4	0.250
blue	$(-1, -2)$	1	1.000
blue	$(2, -2)$	10	0.100
orange	$(-1, 2)$	9	0.111
orange	$(1, 1)$	8	0.125
orange	$(-2, 1)$	9	0.111

Table 2: The list of distances between the red point and each training example from Fig. 1. The vote weight is defined as 1 / distance^2.

The distances^2 between the red point and each training example are presented in Tab. 1. Euclidean metric was used, however, the choice of a metric is arbitrary and Chebyshev distance, cosine similarity etc. can be used as well. Now, lets take a look how the red point is classified for different k:

blue for $k = 1, 2, 3$

orange for $k = 5$

tie for $k = 4, 6$

This was ultra-simple example to demonstrate the method, but still some conclusions can be drawn:

- for binary classification k should be odd to avoid a tie
- large k does not necessary mean better results (especially when $k \sim$ training sets size)
- one can consider simple expansion with *vote weight* depending on the distance

Lets consider once again the given example, but now each point votes for a class membership with the weight given by 1 / distance^2, as in Tab. 2. Now, for any $k \in [1, 6]$ the red point is classified as blue:

k = 1: 1.000 vs 0.000

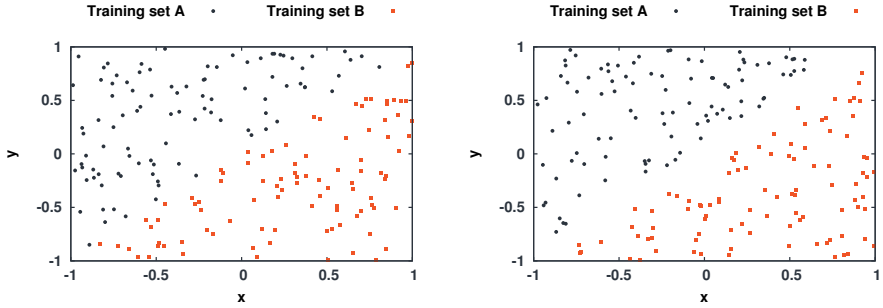
k = 2: 1.250 vs 0.000

k = 3: 1.250 vs 0.125

k = 4: 1.250 vs 0.236

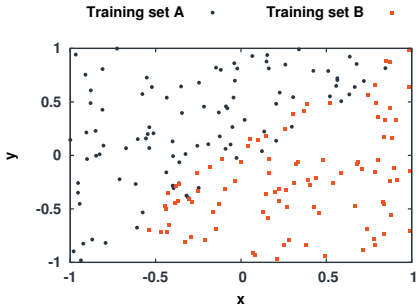
k = 5: 1.250 vs 0.347

k = 6: 1.350 vs 0.347



(a) Default

(b) Distant



(c) Overlapped

Figure 2: Training sets for points separable by $f(x) = x$.

2.1 Separable points

Lets consider something defined mathematically better than *blue* and *orange*, i.e. two classes of points which can be separated linearly, e.g. by the line $f(x) = x$ as presented on Fig. 2 in three scenarios:

- points are exactly separated by $f(x) = x$, Fig. 2(a)
- there is a small gap to make points more separated, Fig. 2(b)
- points overlap a little, Fig. 2(c)

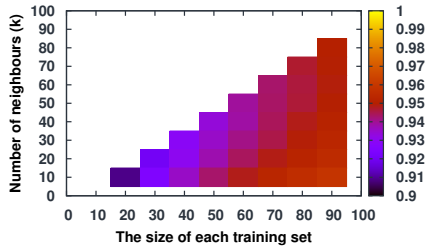
For each scenario the efficiency of kNN as a function of the size of training sets (n) and k is checked in the following way:

1. generate n random points above $f(x) = x$ (training set A)
2. generate n random points below $f(x) = x$ (training set B)
3. generate a random point
4. find k nearest neighbors of this point
5. let them vote:
   - with vote weight = 1 (unweighted)
   - with vote weight = 1/distance (weighted)
6. repeat points 3-5 N times to get good enough statistics ($N = 10^5$ for presented results)
7. calculate the score = no. of correctly guessed points / no. of all test points

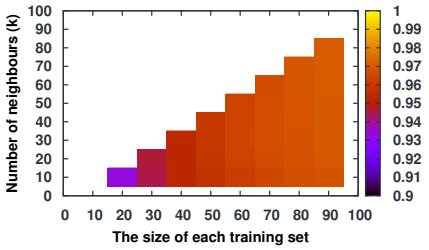
The efficiency of kNN as a function of k and n can be found on Fig. 3. Clearly, weighted voting gives better results, especially for $k \sim n$. However, even unweighted kNN still gives score better than 90%. The efficiency is almost the same when extra gap is generated between training sets (Figs. 3(c) and 3(d)), but it becomes much worse if training sets overlap (Figs. 3(e) and 3(f)).

Lets take a look what happens when training sets are not uniformly distributed, as presented on Fig. 4. It still looks (by eye) that training sets are separated by $f(x) = x$, see Fig. 4(a). However, from obvious reasons some points are going to be incorrectly classified, as presented on Fig. 4(b).

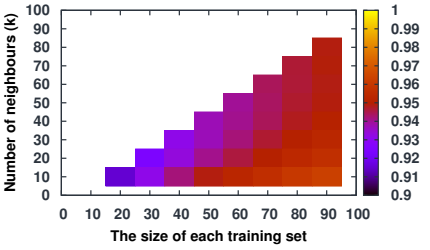
[2] Actually, squares of distances are considered to avoid square roots (it does not affect the result).



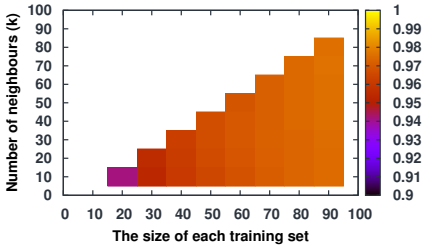
(a) Default, unweighted



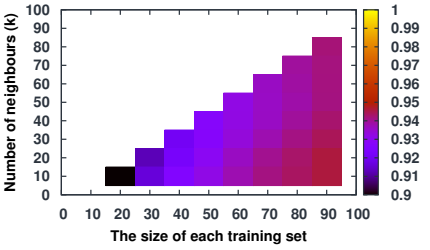
(b) Default, weighted



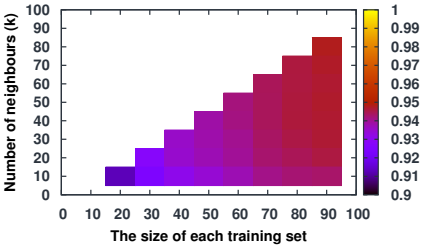
(c) Distant, unweighted



(d) Distant, weighted

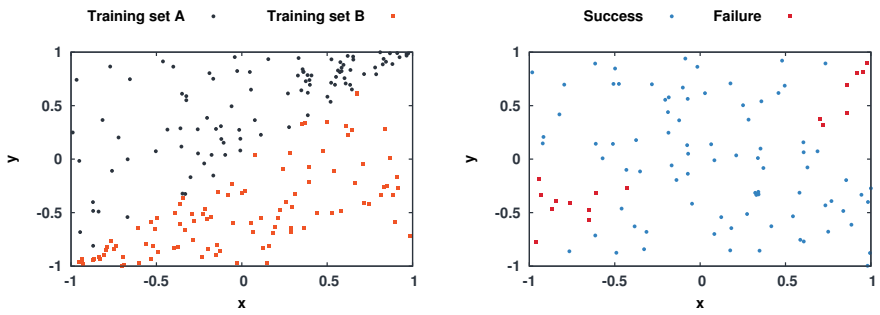


(e) Overlapped, unweighted



(f) Overlapped, weighted

Figure 3: kNN efficiency for points separable by $f(x) = x$.



(a) Training sets (b) Testing set

Figure 4: The example of the kNN classification using wrong training sets ($n = 100$, $k = 50$).