K-Local Hyperplane and Convex Distance Nearest Neighbor Algorithms

Pascal Vincent
http://www.iro.umontreal.ca/vincentp
Dept. IRO,
Universit´e de Montr´eal, Canada
Yoshua Bengio
https://en.wikipedia.org/wiki/Yoshua_Bengio

Generally, there is a similarity between 1NN(1-Nearest Neighbors) and SVM(Support Vector Machines) classifiers when it comes to data which is linearly classified or linearly locally classified. In SVM(support vector machine) we classify data using decision surface/hyperplane, when we extend this to making non-linear decision surfaces, we use kernel trick which maps input to high dimensional space using some non linear transformation. This corresponds to maximizing the margin which is the smallest Euclidean distance from the decision surface to training points in high dimensional space.A different approach is finding a non-linear decision surface, i.e finding the maximum "local margin"(Euclidean distance between a point on decision surface and closest training points) in input space, this is what 1-NN makes! for finite data KNN algorithm is outperformed by SVM in classification tasks, may be due to more capacity of KNN. This is an attempt to answer the question why KNN performance is poor? and to find an update/improvement in the KNN algorithm. As KNN for k=1 assigns nearest point class which is same as finding a local margin, that's why we say solution of KNN algorithm gets closer to the SVM algorithm solution when the data is considered locally linear. For a KNN algorithm, given finite number of data points and some points are missing, the space not covered by these points will deform the decision surface due to random variations in sample density and include some artifacts(holes) in the decision surface, so with regard to unseen(new test points) points close to the local linear manifold will have minimum local margin and have poor performance. But SVM don't have this problem as they make their planes linear. To fix this problem we have to imagine or fill the missing points by taking local linear approximation, which leads to the modified nearest neighbor algorithm.

Here a new algorithm is defined, as in a KNN algorithm distance of a point from another point is calculated, but here the distance of the test point x from the hyperplane is calulated, this hyperplane defined by fantasizing the missing points. These hyperplanes are local affine subspaces and defined for every class c, that passes through K points of K-c neighborhood(K nearest neighbors from the same class c are taken) and has dimensions K-1 or less. given a point x, for every class c, K nearest points are taken and a hyperplane is drawn through those points, which will be a linear plane and called as"local hyperplanes". Formally, this hyperplane is defined as a plane which contains all the K-neighborhood points. So, a plane equation is defined where each point is multiplied with some coefficient α_k.

Equation of the local hyperplane p :

$LH_c^K(x) = \{\ p | p = \Sigma\alpha_k N_k, \alpha \varepsilon R^k, \Sigma\alpha_k = 1\}$

As there is a constraint, we can eliminate it by taking a reference point as origin. So defining centroid as a reference point, N'= $\frac{1}{K}\Sigma N_k$ so revised equation will be:

$LH_c^K(x) = \{\ p | p = N' + \Sigma\alpha_k V_k, \alpha \varepsilon R^k\}$

V_k' is linear basis of difference subspace of (N_k - N').

Hyperplane distance is illustrated as, f(x) = arg min $_\alpha$ d (x, $LH_c^K(x)$).

This distance is named K-Local Hyperplane Distance and hence the name of algorithm is K-Local hyperplane and Nearest Neighbour Algorithm. This distance is calculated for every class :

d(x,$LH_c^K(x)$)= min_p ||x-p||.

substituting the plane equation:

d(x,$LH_c^K(x)$)=min_p ||x -N'-$\Sigma\alpha_k V_k$ ||

this equation can be solved to get α, (V'.V).α = V'.(x-N'), where x and N' are n dimensional column vectors, α is a k dimensional, V = $N_1 - N', N_2 - N', ...N_k - N'$ is a nXK dimensional vector. there are infinite solutions to the α but any solution will work for us.

In HKNN for a given test sample, first step is to find the K points nearest to point in every class, then using these neighbors a hyperplane is constructed and point is associated with class which is closest to the test point in terms of euclidean distance.

This algorithm works similar to Tangent Distance Algorithm, As the hyperplane can be seen as a tangent hyperplane representing the set of points and not affecting them so called invariance. Main difference in HKNN these hyperplanes are derived from the training set (neighborhood of the test point) and in TDA from the prior knowledge. This algorithm also shows some similarities with with Local Learning Algorithm, Local Linear Embedding(for dimensionality reduction), Nearest Feature Line (NFL).

But with this HKNN algorithm there is a problem with large values of K, in high dimensional space the exact colinearities may rare, So any K>n will have different combinations, so we can't construct a hyperplane and sometimes hyperplane will take wrong directions, giving poor performance. so, the fantasizing of the points should be done in small region of hyperplane. two solutions are considered to deal this;

CKNN solution : In this algorithm a convex hull(like envelope surrounding the points) around the K neighbors is defined, which is a subspace of the hyperplane. But to calculate the distance we have to solve a quadratic programming equation, unlike solving a linear one, so this solution is used for the dimension in order of K, this algorithm is referred as CKNN(K-local Convex Distance Nearest Neighbor Algorithm).

The "weight decay" penalty solution: a penalization term is added to the distance function to penalize for large values of α.

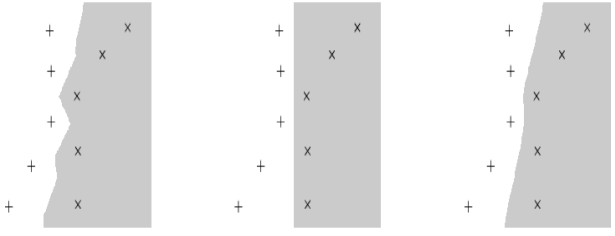
$d(x, LH_c^K(x))^2 = min_p\ ||x - N' - \Sigma\alpha_k V_k||^2 + \lambda\Sigma\alpha_k^2$

the solution for α is now solving the following equation: (V'.V + λI_n).α = V'.(x-N'). This is generilazation of the HKNN algorithm. Here when we add a weight decay term, we have to tune two parameters, λ and K-neighbors(as it is variable here). which we can get if we use separate validation set.

Experimental results suggest that using a KNN alone can't give us good results in case of finite date with missing samples, but HKNN and CKNN shown great improvement in results for many experiments. when implemented on real world classifications tasks like MNIST dataset, HKNN performed upto the level or better than performance of SVM without requiring more data than SVM. the results can be seen below; 1) using USPS and MNIST data ,

Data Set	Algorithm	Test Error	Parameters used
USPS	KNN	4.98%	$K = 1$
(6291 train,	SVM	4.33%	$\sigma = 8, C = 100$
1000 valid.,	HKNN	3.93%	$K = 15, \lambda = 30$
007 test points)	CKNN	3.98%	$K = 20$
MNIST	KNN	2.95%	$K = 3$
(50000 train,	SVM	1.30%	$\sigma = 6.47, C = 100$
10000 valid.,	HKNN	1.26%	$K = 65, \lambda = 10$
)000 test points)	CKNN	1.46%	$K = 70$

In the figure below, 1) KNN is having artifacts in the decision surface(holes are appearing),2)SVM producing a linear hyperplane,3) CKNN produces slightly better surface than KNN.(maximing local margin and smooth surface)



Conclusions : In this paper two modified versions of the KNN are proposed and explained, which shown great improvement in results. HKNN looks encouraging and promising(gives best results when distance is calculated using euclidean metric). HKNN is simple to implement as it is obtained by solving simple equation and adding some steps. These two will have advantages and disadvantages of KNN and their own advantages which makes their performance quite well. Overall, in this paper we can learn new Algorithm and some important theory about KNN.

References(other than reference paper to understand the concepts more):

1) Manifold Based Local Classifiers: Linear and Nonlinear Approaches