A Data Structure for Kernel Nearest Neighbors

Josh Burdick

December 13, 2001

1 Introduction

Text classifiers attempt to sort documents into some set of categories, given a set of example documents from each category. Given an unknown instance, a nearest-neighbor classifier will output the label of the nearest training example. The k-nearest-neighbor rule will output a (possibly weighted) average of the labels of the k nearest examples.

In the last several years, support-vector machines (SVMs) have been successfully applied to many classification tasks, including text classification. They can deal with examples that have large numbers of features, because they represent features indirectly, through *kernel functions*.

In Joachims' comparison of several text classifiers with SVMs, nearest-neighbor performs second only to SVMs. That comparison only used Euclidean distance with the nearest neighbors.

Burges, in his SVM tutorial, points out that the "kernel trick" also can be used with nearest-neighbor classifiers, among others.

Here, we attempt to use several of the kernel functions commonly used with support-vector machines.

Another problem with nearest neighbor classification is implementing it efficiently. One can simply leave the training data in its original form, and do a linear scan through it at classification time. This makes for fast training, but slow classification. Many tree structures have been proposed to speed up classification, at the cost of building some structure at training time. These frequently have trouble with high-dimensional feature spaces.

Here, we use a tree structure based on Voronoi diagrams to approximately answer nearest-neighbor queries, without looking at each training instance. We apply this combination to a text-classification task.

2 Fast Nearest Neighbor searching

2.1 A tree data structure

([?] refers to the use of Voronoi diagrams to speed up nearest-neighbor searching. I couldn't find a description of this, so I'll call the following data structure a Voronoi tree, for lack of a better name. I guess it could also be considered a form of hierarchical clustering.)

We will build a tree with w points at each node. Each node is associated with a subtree. To add a point x to the tree, if a given node n has fewer than w points, add x to this node. Otherwise, recurse, by adding x to the subtree associated with the point nearest x in n.

Thus, at each node, this tree implicitly constructs a Voronoi diagram. We guess that perhaps the nearest neighbor to x will be in the region whose center is closest to x; or, if it's not there, perhaps it's in the second-nearest region, etc. This isn't actually true; there are counterexamples in which the nearest neighbor isn't in the nearest $w/2$ regions.

We would expect to be able to trade off speed for accuracy, by only searching the nearest s subregions at each node. Unfortunately, if d is the depth of the tree, then we need to search s^d nodes, which is exponential in s. We can counteract this by increasing w, but then we need to do more distance calculations at each node.

It's not obvious what the best tradeoff here is, especially lacking any (even approximate or probabilistic) bound on the various parameters' effect on accuracy.

Note that this structure doesn't explicitly mention the dimensionality of the problem (although the dimensionality certainly could affect its performance.)

3 Kernels

Any kernel which can be used for SVMs can also be used with nearest neighbors. We are given a feature space X, a (presumably higher-dimensional) feature space Z, and a transformation $\theta : X \to Z$. Given two points $x_1, x_2 \in X$, we are interested in finding the distance between their images, $\theta(x_1), \theta(x_2) \in Z$.

Suppose we have a kernel function $K : (Z, Z) \to R$, such that $K(x_1, x_2) = \theta(x_1) \cdot \theta(x_2)$. Then we can write the square of their distance as $\theta(x_1 - x_2) \cdot \theta(x_1 - x_2)$.

We can rewrite this

$$\begin{aligned} & \theta(x_1 - x_2) \cdot \theta(x_1 - x_2) \\ = \theta(x_1) \cdot \theta(x_1) - 2\theta(x_1) \cdot \theta(x_2) + \theta(x_2) \cdot \theta(x_2) \\ = K(x_1, k_1) - 2K(x_1, x_2) + K(x_2, x_2) \end{aligned}$$

to a function which only refers to the x_1 and x_2 through K. [Lewis]

4 Results

We first attempted to see empirically how good an approximation the Voronoi tree is returning. Then, we measured how well a classifier based on a Voronoi tree could classify text documents.

All experiments were performed on a Pentium (?Mhz) with 128Mb of memory, using Sun's JDK 1.3.1 Java implementation. The tests of the approximation performance were run under Windows NT, and the text classification experiments were run under Linux 2.2.

4.1 Approximation performance

Before measuring classification performance, we attempted to measure just the error in the Voronoi trees' approximation to nearest neighbor. To do this, we selected 1000 points uniformly from $[0, 1]$, and built a tree from them, with $w = 16$ subtrees at each node. Then, for each of another 100 points uniformly chosen from $[0, 1]$, we measured the *approximation ratio r*

(distance to nearest neighbor given by the tree) /

(actual distance to nearest neighbor)

For this experiment, we only used the Euclidean distance metric, both in building the tree, and searching it. There was a bug in the code, because when the search width was set to search the entire tree, the approximation ratio was not 1. However, it still seemed to produce reasonable approximations.

dimensionality	search width	approximation ratio	standard deviation
5	1	1.270	0.501
50	1	1.096	0.072
500	1	1.031	0.163
5	2	1.055	0.168
50	2	1.060	0.054
500	2	1.022	0.014
5	4	1.032	0.191
50	4	1.037	0.044
500	4	1.014	0.013
5	8	1.008	0.041
50	8	1.019	0.033
500	8	1.006	0.009

As expected, we get improved accuracy when we increase the search width. Note that, in some cases, the approximation ratio improved as the dimensionality increased. This is odd.

4.2 Classification performance

We tested classification accuracy on a subset of the "20 newsgroups" corpus (`rec.autos`, `rec.motorcycles`, `rec.sport.baseball`, `rec.sport.hockey`.)

We used the bag-of-words model. Documents were represented by a vector of word counts, which was normalized to 1. Words in the SMART stoplist were discarded. We used the polynomial kernel, and varied the polynomial degree and the search width.

We tested on five random partitions of the data.

polynomial degree	search width	accuracy	standard deviation
*	1	84.5%	0.012
*	3	90.1%	0.02

It appears that there was a bug here, as well, because varying the polynomial degree left the performance well within 1 standard deviation of what's reported above.

The accuracies obtained are around the weighted k-NN accuracies obtained by Joachims, not using kernels. Unfortunately, those experiments used weighted k-NN, and a different corpus, and so those results aren't directly comparable.

5 Conclusions

We have described a tree structure to approximately find nearest neighbors. The tree structure makes no explicit mention of the dimensionality of the feature space. Although we have no theoretical approximation or probabilistic bounds on its performance, we find that it gives respectable performance in practice. We use that tree structure to classify text, and find it performs respectably for a nearest-neighbor classifier.

This classifier is probably faster to train than SVMs currently are, but somewhat slower at classification time. This isn't a terribly useful niche, though – for most applications, one is happy to put up with slower training, in exchange for faster classification.

This study didn't compare performance with other learning methods; such a comparison would be interesting.

6 Acknowledgements

Thanks to Fernando Pereira for help with the kernel distance measure, and to Andrew Schein and Eugene Buehler for suggesting that trying kernel methods with nearest neighbor would be an interesting project.

7 References

C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, volume 2, number 2, pp. 121-167, 1998.

C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273-297, November 1995.

T. Joachims. Text categorization with Support Vector Machines: Learning with many relevant features. In *Machine Learning: ECML-98, Tenth European Conference on Machine Learning*, pp. 137–142.

Lewis, David. Description of the KKNN method and p-value computation. Available on the Web at `http://www.cs.columbia.edu/ dplewis/research/kknn/`