ECS 171 - Forest Cover Project Write-up: Pine Subgroup

Ian Woods, Jesse Dyer, Miguel Covarrubias

December 14, 2014

Brief Description of KNN

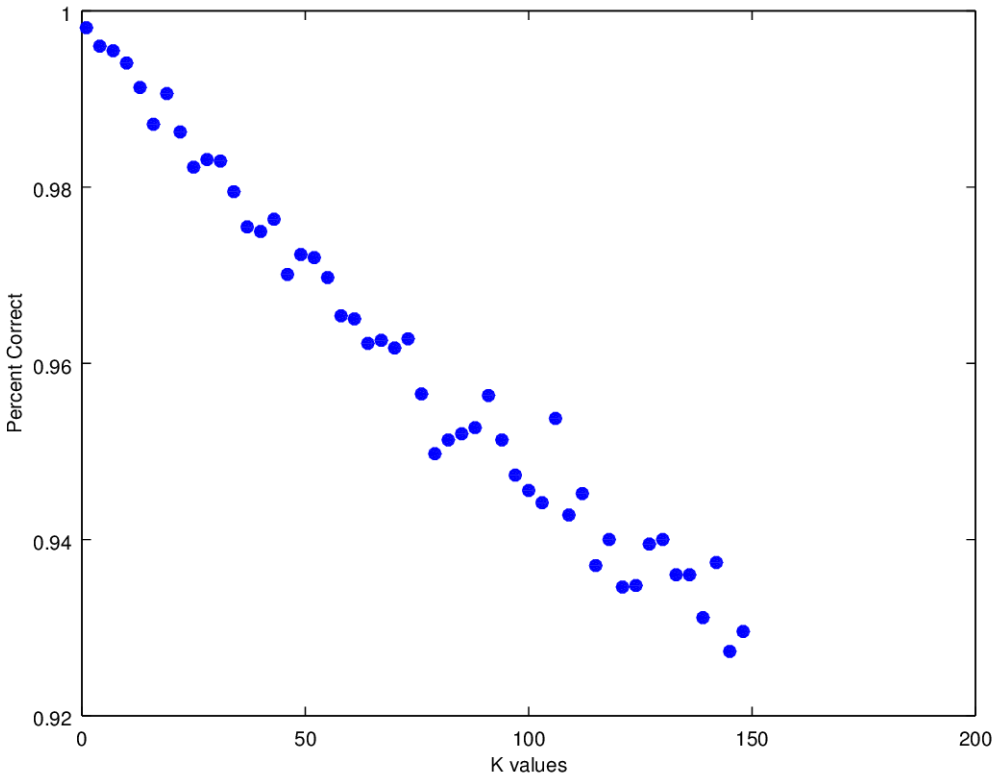
The k-Nearest Neighbors (KNN) Algorithm involves classifying a given sample as a particular type, given the type of the k closest samples to it, as determined by some distance function. Our initial approach was to use a standard Euclidean distance function, defined as follows:

$$\sqrt{\sum_{k=1}^{n}(x_{jk} - x_{ik})^2}$$

We decided to begin with this naive approach to isolate failure points (essentially, if something were to work incorrectly, we could rule out the distance function as failure point). We debated creating a distance function tailored to the dataset by hand, or by simply trying many different possible distance functions. However, we decided to generate a distance function with the largest-margin nearest neighbors algorithm (more precisely, we generated a transformation matrix that was used to transform the data, and still used the Euclidean distance).

Using different values for k can theoretically alter the predicted results - for example, if a sample of unknown type is closest to a sample of type A with a calculated distance of 1, but two samples of type B are at a distance of 1.1 from our unknown sample, choosing $k = 1$ will yield the result A for the unknown's type, and $k = 3$ will yield the result B. We decided to test a wide range of k values.

Results of Standard KNN Algorithm for Various K Values



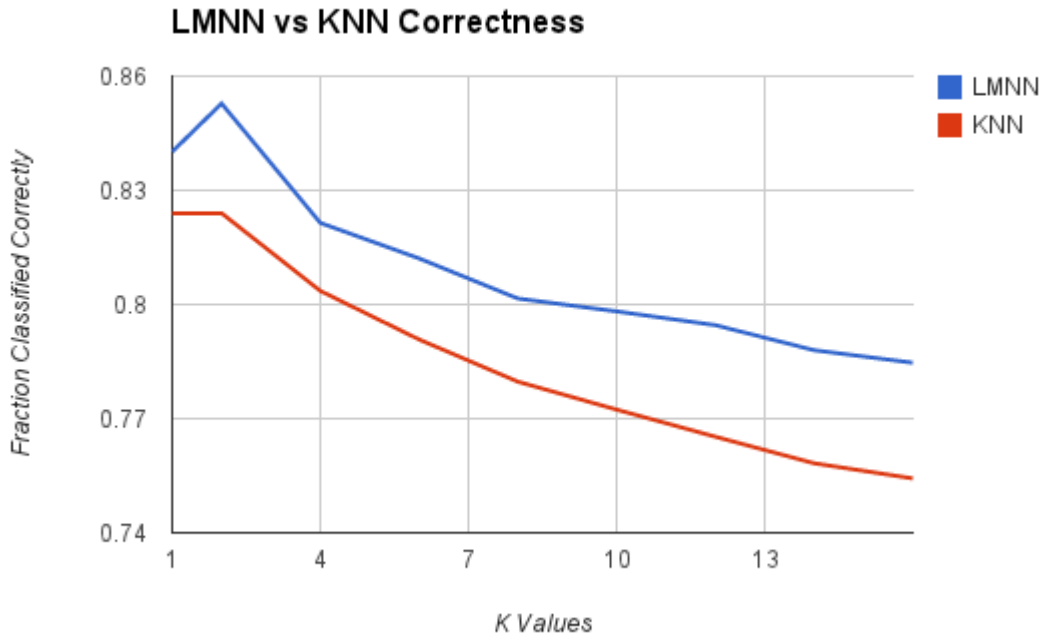
As can be seen in the above graph, increasing the value of k reduced the accuracy of the classifier. We tested over a range from 1 to 150 with a step size of 3 (to reduce computation time while still revealing the overall trend). This makes intuitive sense - imagine a KNN implementation with $k =$ the size of the data set. Say there are n samples of the most common type, and z total samples. Every sample would be labeled as the most common sample, resulting in an accuracy of $\frac{n}{z}$.

Brief Description of LMNN

LMNN (Largest Margin Nearest Neighbors) is a variant of KNN that can be implemented in two ways. One method is to learn a Mahalanobis distance

function, which measures the distance (in standard deviations) from the center of a cluster of similarly typed samples. Another method, and the method our implementation used, is to learn a transformation matrix that, when applied to the dataset, yields higher correct classification rates. It learns this matrix by iteratively applying small perturbations to the transformation matrix with the goal of maintaining a large distance between "impostors" (close neighbors that are of the wrong type) and the perimeters that surround groups of similarly-typed neighbors. LMNN is very helpful because it saves use the trouble of testing many different distance functions, or trying to derive a distance function by hand for each particular dataset. Pictured below are the results of our LMNN classifier, compared to the standard KNN classifier.

Results of LMNN



LMNN was able to boost the correct classification percentage in every case. On average, we saw 2.46% better classification rates. It took quite a lot of extra time to learn the distance function, though - running the LMNN algorithm on the entire UCI dataset took on the order of one hour on a

relatively high-powered desktop computer (quad core Intel i5 processor at 3.4GHz, all CPUs at 95% + load for most of the computation).
Below are the ROC and PR curves for our LMNN classifier for each of the 7 forest cover types.