

CS 374 Assignment #7

Instance-Based Learning Algorithms: k-Nearest Neighbor

Due the week of March 31, 2008

All of the learning techniques we have studied, whether for classification or regression tasks, have involved the analysis of a training set of examples in order to build a model for the task at hand. New examples were then analyzed using the learned model. This week we consider the k-nearest neighbor algorithm, which does not explicitly build a model from training data. Instead, it stores the training instances so that they can be used to analyze new instances later.

As its name implies, the k-nearest neighbor algorithm searches the training examples to find those that are “closest” to a new example to be analyzed. It then uses these to determine the appropriate output for the new instance. The k-nearest neighbor algorithm is an example of a class of learning techniques called instance-based methods. These methods are sometimes referred to as “lazy”, since they put off the processing of examples until a new test instance must be analyzed.

This will be the last in our series of “top algorithms for classification and regression” for a while.

1 k-Nearest Neighbor Learning

1.1 Reading

This week you should begin by reading

- Mitchell, pages 230-236.

This gives a very nice introduction to the k-nearest neighbor algorithm, with a focus on real-valued attributes. It discusses the advantages and disadvantages of this learning approach and outlines solutions for handling some of the problems. For example, it points out that not all neighbors of a new instance should be weighted equally in determining the output value for the new instance. It also points out that the process of finding the nearest neighbors of a new instance can be costly.

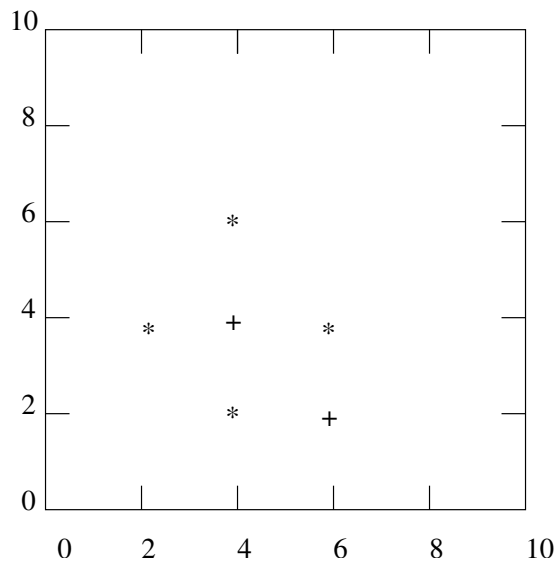
Next you should read

- Russell and Norvig, pages 733-735, and
- Witten and Frank, pages 78-79 and pages 128-135.

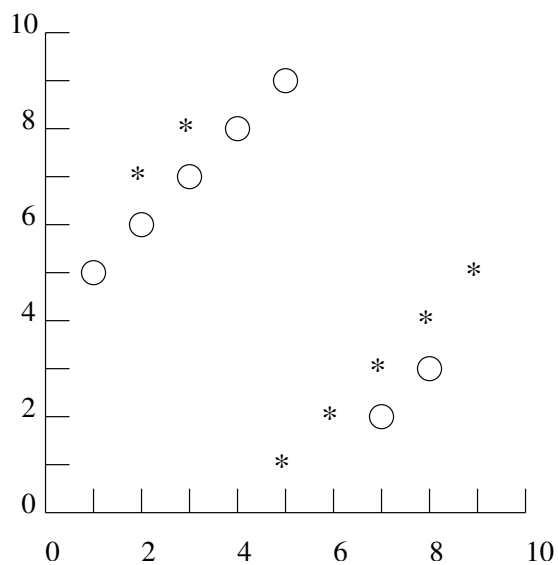
The reading in Russell and Norvig provides alternate ways to determine nearest neighbors, as does the reading in Witten and Frank. (Both, for instance, discuss how we handle nominal, i.e., discrete, attributes.) The Witten and Frank reading also describes (at a very high level) two data structures that can be used for storing training instances so that nearest neighbors of a new instance can be efficiently computed.

1.2 Exercises

1. (Adapted from Dietterich) Consider the set of training examples in the diagram below.



- (a) Draw the decision boundaries for the 1-nearest neighbor algorithm assuming that we are using standard Euclidean distance to compute nearest neighbors. A plus indicates a positive example and a star indicates a negative example.
- (b) How will the point (8, 1) be classified by the 1-nearest neighbor classifier?
- (c) How will the point (8, 8) be classified?
2. (Modified from Mitchell and Guestrin) One of the problems with k-nearest neighbor learning is selecting a value for k. For this exercise, you will use Weka to empirically determine a reasonable value for k, given a specific training set.
- Say you are given the data set shown below. This is a binary classification task in which the instances are described by two real-valued attributes.



- (a) What value of k minimizes training set error for this data set, and what is the resulting training set error? (See part d of this exercise for information on how to obtain the data set and how to select the k -nearest neighbor classifier in weka, though you shouldn't need to use it here.)
- (b) Why is training set error not a reasonable estimate of test set error, especially given the value of k chosen in part a)?
- (c) Why might using too large a value of k be bad for this data set? Why might using too small a value be bad for this data set?
- (d) Now use Weka to help you find a value of k that minimizes leave-one-out cross-validation error. What we'll be doing here is using our training set to help us select a good value for k . The idea of using leave-one-out cross-validation should be a familiar one (recall the back-propagation assignment). In using it to select k , we do the following. We take our training instances and divide them into two groups, so that some are used for training and others serve as validation (i.e., "test") examples. Specifically, for the given training set of 14 examples, we divide it into a set of 13 for training and a set of 1 for testing. There are obviously 14 ways to choose 13 ("leave one out"), and we will consider all 14.

For each training-validation split of the data (remember that there are 14 of these), we run the k -nearest neighbor algorithm for all possible values of k (1 to 14). We then compute how well we did at classifying the validation instances for each value of k . The k that performs best is selected as the value to use for future (unseen) test instances.

To perform this experiment, you need to begin by copying the data file `week7.arff`, which can be found in

```
~andrea/shared/cs374
```

Then start up Weka as you did last week, by typing

```
java -jar weka.jar
```

This starts up the Weka GUI. Click on the "Experimenter" button. This should open a window for the Weka Experiment Environment. Here you can set up an experiment like the one described above.

First click to select "Simple" experiment, and then click "New" to set up a new experiment. Next you should type in the name of a file in which you would like to store the details of the experiment. Now under "Experiment Type", select "Cross-validation" and specify the number of folds to be 14. Also be sure that you've selected "Classification" under "Experiment Type."

The next thing you need to do is select the data set. In this case there's only one - `week7.arff`.

Under "Iteration Control", set the number of repetitions to 1, and select "Data sets first". Finally, you need to specify the algorithms to be run, i.e., 1-nn, 2-nn, 3-nn, ..., 14-nn. Click on "Add new", and then click on "Choose". In the "lazy" folder, select "IBk" (i.e., instance based k). Now you can edit the specifics of the algorithm. First make "KNN" 1. You should set "crossValidate" to False and "noNormalization" to true. You will need to do this 14 times.

Now you're ready to run the experiment. Click on "Run" at the top of the window. The experiment won't actually run until you click "Start", so do that now.

Finally, you can "Analyse" the results. To do this, click on "Analyse", and then click on "Experiment", which you'll find near the top right hand side of the window. The results of

the experiment can be presented in many different ways. What you want is to see either the “Number correct” or the “Percent correct”. You can choose this by selecting the “Comparison field”. To actually see the results displayed in this way, click on “Perform test.”

What value of k minimizes leave-one-out cross-validation error for this data set?

Play around with this interface for a bit, as you will be performing more experiments in the coming weeks.

3. (From Mitchell and Guestrin) A well-known result by Cover and Hart (1967) is that the asymptotic error rate of the 1-nearest neighbor classifier is at most twice the Bayes-optimal error rate. In this problem, you will prove Cover and Hart’s theorem for the case of binary classification with real-valued attributes.

Let x_1, x_2, \dots be training instances (points in d -dimensional Euclidean space, for some fixed d , and let y_1, y_2, \dots be the corresponding class labels ($y_i \in \{0, 1\}$). Let $p_i(x) = p(X = x | Y = i)$ be the conditional probability distribution for points in class i ; we assume $p(x) > 0$ for all x . Let $R = p(Y = 1)$ be the probability that a randomly generated point is in class 1; we assume $0 < R < 1$.

- (a) From these expressions, we can easily calculate the true probability $q(x) = p(Y = 1 | X = x)$ that any data point x was generated by class 1. Express $q(x)$ in terms of $p_0(x)$, $p_1(x)$, and R .

From this expression for $q(x)$, it is clear that $q(x)$ is defined for all x , and $0 < q(x) < 1$.

- (b) A Bayes-optimal classifier is a classifier that always assigns a data point x to the most probable class, $\operatorname{argmax}_y P(Y = y | X = x)$, thus minimizing the value of the 0-1 loss function, or equivalently, maximizing the probability of correct classification. Given some test point x , what is the expected error of the Bayes-optimal classifier, in terms of $q(x)$?
- (c) The 1-nearest neighbor classifier assigns a test data point x the label of the closest training point x' . Given some test data point x with nearest neighbor x' , what is the expected error of the 1-nearest neighbor classifier, in terms of $q(x)$ and $q(x')$?
- (d) In the asymptotic case, the number of training examples of each class goes to infinity, and the training data fills the space in a dense fashion. As a result, the nearest neighbor to x has $q(x')$ converging to $q(x)$. By performing this substitution in the previous expression, give the asymptotic error for the 1-nearest neighbor classifier at point x , in terms of $q(x)$.
- (e) Show that the asymptotic error obtained in part d) is less than twice the Bayes-optimal error obtained in part b).

2 Locally Weighted Regression

In week 3 you read a paper on applying Naive Bayes to regression. The authors of that paper compared their algorithm to others, including one that did locally weighted regression. Locally weighted regression makes use of a nearest neighbor approach to find those training instances closest to a new test instance. It then constructs a model of the target function with a focus on the points in the local neighborhood of the test instance. Now that you have learned about k -nearest neighbor learning, you can gain a better understanding of locally weighted regression.

2.1 Reading

Please read

- Mitchell, pages 236-238.

Though there are many details omitted from the discussion, these pages will give you a good overall sense of the algorithm. (If you're interested in a much more detailed explanation, you can read the paper "Locally Weighted Regression" by Atkeson, Moore, and Schaal. A link to this paper can be found on the Assignments web page for this course.)

The reading in Mitchell emphasizes locally weighted *linear* regression. As you know, there are many ways to learn a linear function, including the linear regression algorithm we considered in week 4 and gradient descent as we discussed in week 1. Mitchell discusses the latter, which provides a wonderful opportunity for revisiting this earlier topic.

2.2 Exercise

1. (Modified from Mitchell) Derive the gradient descent rule for a distance-weighted local linear approximation to the target function, given by Equation (8.7).