Nearest Neighbor Classification

Charles Elkan
elkan@cs.ucsd.edu

January 11, 2011

What is called supervised learning is the most fundamental task in machine learning. In supervised learning, we have training examples and test examples. A training example is an ordered pair $\langle x, y \rangle$ where x is an instance and y is a label. A test example is an instance x with unknown label. The goal is to predict labels for test examples. The name "supervised" comes from the fact that some supervisor or teacher has provided explicit labels for training examples.

Assume that instances x are members of the set X, while labels y are members of the set Y. Then a classifier is any function $f : X \to Y$. A supervised learning algorithm is not a classifier. Instead, it is an algorithm whose output is a classifier. Mathematically, a supervised learning algorithm is a higher-order function: it is a function of type $(X \times Y)^n \to (X \to Y)$ where n is the cardinality of the training set.

The nearest-neighbor method is perhaps the simplest of all algorithms for predicting the class of a test example. The training phase is trivial: simply store every training example, with its label. To make a prediction for a test example, first compute its distance to every training example. Then, keep the k closest training examples, where $k \geq 1$ is a fixed integer. Look for the label that is most common among these examples. This label is the prediction for this test example. Using the same set notation as above, the nearest-neighbor method is a function of type $(X \times Y)^n \times X \to Y$. A distance function has type $X \times X \to \mathbb{R}$.

This basic method is called the kNN algorithm. There are two major design choices to make: the value of k, and the distance function to use. When there are two alternative classes, in order to avoid ties the most common choice for k is a small odd integer, for example $k = 3$. If there are more than two classes, then ties are possible even when k is odd. Ties can also arise when two distance values are

the same. An implementation of kNN needs a sensible algorithm to break ties; there is no consensus on the best way to do this.

When each example is a fixed-length vector of real numbers, the most common distance function is Euclidean distance:

$$d(x, y) = ||x - y|| = \sqrt{(x - y) \cdot (x - y)} = (\sum_{i=1}^{m} (x_i - y_i)^2)^{1/2}$$

where x and y are points in $X = \mathbb{R}^m$.

An obvious disadvantage of the kNN method is the time complexity of making predictions. Suppose that there are n training examples in $\mathbb{R}^m$. Then applying the method to one test example requires $O(nm)$ time, compared to just $O(m)$ time to apply a linear classifier such as a perceptron. If the training data are stored in a sophisticated data structure, for example a kd-tree, then finding nearest neighbors can be done much faster if the dimensionality m is small. However, for dimensionality $m \geq 20$ about, no data structure is known that is useful in practice [Kibriya and Frank, 2007].

1 Categorization of supervised learning tasks

A major advantage of the kNN method is that it can be used to predict labels of any type. Suppose that training and test examples belong to some set X, while labels belong to some set Y. As mentioned above, formally a classifier is a function $X \to Y$. Supervised learning problems can be categorized as follows.

- The simplest case is $|Y| = 2$; the task is a binary classifier learning problem.
- If Y is finite and discrete, with $|Y| \geq 3$, the task is called a multiclass learning problem.
- If $Y = \mathbb{R}$ then the task is called regression.
- If $Y = \mathbb{R}^q$ with $q \geq 1$ then the task is called multidimensional regression. Note that the word "multivariate" typically refers to the fact that the dimensionality of input examples is $m > 1$.
- If Y is a powerset, that is $Y = 2^Z$ for some finite discrete set Z, then the task is called multilabel learning. For example Y might be the set of all newspaper articles and Z might be the set of labels {SPORTS, POLITICS,

BUSINESS, ...}. Technically, the label of an example is a set, and one should use a different name such as "tag" for each component of a label.

- If $X = A^*$ and $Y = B^*$ where A and B are sets of symbols, then we have a sequence labeling problem. For example, A^* might be the set of all English sentences, and B^* might be the set of part-of-speech sequences such as NOUN VERB ADJECTIVE.

Tasks such as the last two above are called structured prediction problems. The word "structured" refers to the fact that there are internal patterns in each output label. For example, suppose that x is a sentence of length 2. The part-of-speech labels $y =$ NOUN VERB and $y =$ VERB NOUN both have high probability, while the labels $y =$ NOUN NOUN and $y =$ VERB VERB do not. One major theme of recent research in machine learning has been to develop methods that can learn and use structure in output labels.

The nearest neighbor method is the only algorithm that is directly usable for all the types of problem listed above. Of course, we still need a useful distance function $d : X \times X \to \mathbb{R}$, which may not be obvious to design for some sets X. Also, a kNN method with $k \geq 2$ requires some sort of averaging or voting function for combining the labels of multiple training examples, which may also not be obvious to design.

2 Nearest neighbors in high dimensions

The nearest-neighbor method suffers severely from what is called the "curse of dimensionality." This curse has multiple aspects. Computationally, as mentioned above, if the dimensionality $m \geq 20$ the method is very slow. More subtly, the accuracy of the method tends to deteriorate as m increases. The reason is that in a high-dimensional space all points tend to be far away from each other, so nearest neighbors are not meaningfully similar. Practically, if examples are represented using many features, then every pair of examples will likely disagree on many features, so it will be rather arbitrary which examples are closest to each other.

In many domains, data are represented as points in a high-dimensional space, but for domain-specific reasons almost all points are located close to a subspace of low dimensionality. In this case the "effective dimensionality" of the data is said to be small, and nearest-neighbor methods may work well.

A useful way to judge the impact of the curse of dimensionality is to plot the distribution of interpair distances in the training set. If n is large it is enough to

take a random sample of all $n(n-1)/2$ pairs. It is desirable for the distribution to have large spread compared to its mean. If it does, then the effective dimensionality of the data is small.

For examples that are real-valued vectors using a p-norm distance with $p < 2$ instead of Euclidean distance can be beneficial [Aggarwal et al., 2001]. This distance function is

$$d_p(x, y) = ||x - y||_p = (\sum_{i=1}^{m} |x_i - y_i|^p)^{1/p}.$$

Note that the definition includes taking the absolute value of $x_i - y_i$ before raising it to the power p. The case $p = 1$ is called Manhattan distance: $d_1(x, y) = \sum_{i=i}^{m} |x_i - y_i|$. The case $p = 0$ is called Hamming distance:

$$d_0(x, y) = \sum_{i=1}^{m} I(x_i \neq y_i)$$

where $I(\alpha)$ is an indicator function: $I(\alpha) = 1$ if α is true and $I(\alpha) = 0$ otherwise.

3 Theoretical results

The nearest-neighbor method has an important guarantee. First we need a definition. The Bayes error rate for a classification problem is the minimum achievable error rate, i.e. the error rate of the best possible classifier. This error rate will be nonzero if the classes overlap. For example, suppose that $x \in \mathbb{R}$ and x given i, where i is the class label of x, follows a Gaussian distribution with mean μ_i and fixed variance. The two Gaussians overlap so no classifier can predict the label i correctly for all x, and the Bayes error rate is nonzero. (Note that the Bayes error rate has no connection with Bayes' rule.)

The Bayes error rate is the average over the space of all examples of the minimum error probability for each example. The optimal prediction for any example x is the label that has highest probability given x. The error probability for this example is then one minus the probability of this label. Formally, the Bayes error rate is

$$E^* = \int_{x \in X} p(x)[1 - \max_i p(i|x)]$$

where the maximum is over the c possible labels $i = 1$ to $i = c$.

The theoretical property of the 1NN method is that in the limit where the number of training examples tends to infinity, the error rate of a 1NN classifier is at worst twice the Bayes error rate. This result was proved by Cover and Hart in 1967. For a sketch of the proof see [Ripley, 1996, page 192].

Theorem: For sufficiently large training set size n, the error rate of the 1NN classifier is less than twice the Bayes error rate.
Proof: Let x be a query point and let r be its closest neighbor. The expected error rate of the 1NN classifier is

$$\sum_{i=1}^{c} p(i|x)[1 - p(i|r)]$$

where $p(i|x)$ is the probability that x has label i and $1 - p(i|r)$ is the probability that r has a different label. The critical fact is that if the number n of training examples is large enough, then the label probability distributions for all x and r will be essentially the same. In this case, the expected error rate of the 1NN classifier is

$$\sum_{i=1}^{c} p(i|x)[1 - p(i|x)].$$

To prove the theorem we need to show that

$$\sum_{i=1}^{c} p(i|x)[1 - p(i|x)] \;\leq\; 2[1 - \max_i p(i|x)].$$

Let $\max_i p(i|x) = r$ and let this maximum be attained with $i = j$. Then the lefthand side is

$$r(1 - r) + \sum_{i \neq j} p(i|x)[1 - p(i|x)]$$

and the righthand side is $2(1 - r)$. The summation above is maximized when all values $p(i|x)$ are equal for $i \neq j$. The value of the lefthand side is then

$$\begin{aligned} A &= r(1 - r) + (c - 1)\frac{1 - r}{c - 1}\frac{(c - 1) - (1 - r)}{c - 1} \\ &= r(1 - r) + (1 - r)\frac{c + r - 2}{m - 1}. \end{aligned}$$

Now $r \leq 1$ and $c - 2 + r < c - 1$ so $A < 2(1 - r)$ which is what we wanted to prove. ■

An alternative version of this result is the statement $E^* \geq E/2$. This says that with a large enough training set, no classifier can do better than half the error rate of a 1NN classifier. Conversely, we can obtain an estimated lower bound for the Bayes error rate by measuring the error rate of a 1NN classifier, then dividing by two.

Can a nearest-neighbor method actually come close to the Bayes error rate? The answer is yes. If $k \to \infty$ while $k/n \to 0$ then the error rate of kNN converges to the Bayes error rate as $n \to \infty$. The result is strongest with $k/\log n \to \infty$ also [Devroye et al., 1994]. Note that because these results are for $n \to \infty$ they do not say which k is best for any particular finite n.

The three results above are true regardless of which distance metric is used. If the sample size is large enough, then any distance metric is adequate. Of course, for reasonable finite sample sizes, different distance metrics typically give very different error rates.

4 The LAESA algorithm

If the distance function satisfies the triangle inequality, then this inequality can be used to reduce the number of distance calculations needed. The fundamental fact that is useful is the following.

Lemma: Suppose $d(\cdot, \cdot)$ is a distance metric that satisfies the triangle inequality, meaning that $d(x, z) \leq d(x, y) + d(y, z)$ for all x, y, and z. Then for all u, v, and w $d(u, w) \geq |d(u, v) - d(v, w)|$.

Proof: Note that $d(x, z) - d(y, z) \leq d(x, y)$ and similarly, $d(y, z) - d(x, z) \leq d(x, y)$. Hence $|d(x, z) - d(y, z)| \leq d(x, y)$. ■

The LAESA algorithm applies the lemma above as follows. Given a training set T and a test point x, the algorithm returns the nearest neighbor y in T of x. The algorithm uses a fixed set of basis points B whose distances to every point in T are computed in advance. For any test point x:

compute $d(b, x)$ for all $b \in B$
for all $t \in T$
    compute lower bound $g(t, x) = \max_b |d(t, b) - d(b, x)|$
// loop invariant: $d(t, x) \geq g(t, x)$ for $t \in T$
initialize $y = \text{argmin}_b\ d(b, x)$
for all r in T sorted by increasing $g(r, x)$
    if $g(r, x) \geq d(y, x)$ then return y and finish

else compute $d(r, x)$
if $d(r, x) < d(y, x)$ then set $y = r$

Distances are computed just once between all training points and all points in the basis set B, in a preprocessing stage. The cost of this preprocessing is amortized over many test points x. The algorithm avoids computing distances by doing bookkeeping calculations involving lower bounds that are scalars. The relative computational benefit is larger when distance calculations are more expensive. This is true for Euclidean distance in high dimensions, and for distance functions that are not Euclidean but satisfy the triangle inequality, such as edit distance between strings. For Euclidean distance in low dimensions, there may be little benefit.

5 Learning a similarity function

Suppose that examples are points in Euclidean space $\mathbb{R}^m$. The standard distance function is

$$d(x, y) = (\sum_{i=1}^{m} (x_i - y_i)^2)^{1/2}.$$

This function implicitly places equal emphasis on every dimension i. Intuitively, some dimensions may be more important than others, so they should have higher weight. Also, if some dimensions have a numerical range that is larger than that of others, then giving dimensions equal emphasis may require unequal weights. This point of view leads to considering a weighted distance function

$$d(x, y) = (\sum_{i=1}^{m} w_i (x_i - y_i)^2)^{1/2}$$

where the weight vector $w \in \mathbb{R}^m$ should be learned.

Rather than learn weights for a distance function, consider a mathematically simpler formulation. Define the weighted similarity of two points x and y to be

$$s(x, y; w) = \sum_{i=1}^{m} w_i (x_i y_i)$$

where the semicolon notation indicates that x and y are inputs while w is a parameter. Note that before applying this similarity function, it may be useful to scale

every data point so that it has the same magnitude, for reasons explained in the answer to a quiz question below. A more general formulation is

$$s(x, y; w) = \sum_{i=1}^{m} \sum_{j=1}^{m} w_{ij} x_i y_j.$$

This formulation allows similarity to depend on the interaction between every component of the vector x and every component of the vector y.

A heuristic way to learn weights is to create a training set as follows. For two points that should be similar, make a training example with label +1. For two points that should be dissimilar, make a training example with label -1, or alternatively 0. In both cases, the instance is the vector

$$xy' = \langle x_1 y_1, \ldots, x_i y_j, \ldots, x_m y_m \rangle = T.$$

Above, T is a matrix of dimension $m \times m$ that is called the outer product of the vectors x and y. The notation xy' refers to matrix multiplication. In machine learning, a data point x is usually viewed as a row vector. However, in mathematics it is usual to assume that vectors are columns vectors. The notation y' means the transpose of y, which is a row vector. Note that $x'y$ is the dot product $x \cdot y$, which is a scalar, that is, a single real number.

When using the approach above to learn a similarity function, there are a quadratic number $O(n^2)$ of possible training examples, and a quadratic number m^2 of parameters w_{ij} to learn. Both these numbers may be larger than feasible. To reduce the number of training examples, it is reasonable to take a sample. To reduce the effective number of parameters, the standard approach is called regularization. Regularization is discussed in the lecture notes on linear regression.

6 Linear regression

In this section we discuss linear regression, both because it is useful for learning similarity functions, and because it is a preview of some of the most fundamental general concepts of machine learning: linear model, loss function, regularization, etc. Note the change in notation below compared to the previous section.

Let x be an instance in $\mathbb{R}^p$ and let y be its real-valued label. Write $x = \langle x_1, x_2, \ldots, x_p \rangle$. The linear model is

$$y = b_0 + b_1 x_1 + b_2 x_2 + \ldots + b_p x_p.$$

The righthand side above is called a linear function of x. The linear function is defined by its coefficients b_0 to b_p. The coefficient b_0 is called the intercept. It is the value of y predicted by the model when $x_i = 0$ for all i.

Suppose that the training set has cardinality n, i.e. it consists of n examples of the form $\langle x_i, y_i \rangle$, where $x_i = \langle x_{i1}, \ldots, x_{ip} \rangle$. Let b be any set of coefficients. The predicted value for x_i is

$$\hat{y}_i = f(x_i, b) = b_0 + \sum_{j=1}^{p} b_j x_{ij}.$$

If we define $x_{i0} = 1$ for every i, then we can write

$$\hat{y}_i = \sum_{j=0}^{p} b_j x_{ij}.$$

Finding good values for the coefficients b_0 to b_p is the job of the training algorithm. The standard approach is to choose values that minimize the sum of errors on the training set, where the error on training example i is defined to be the square $(y_i - \hat{y}_i)^2$. The training algorithm then finds

$$\hat{b} = \text{argmin}_b \sum_{i=1}^{n} (y_i - \sum_{j=0}^{p} b_j x_{ij})^2.$$

The objective function $\sum_i (y_i - \sum_j b_j x_{ij})^2$ is called the sum of squared errors, or SSE for short.

Generally, the function that defines the error of a prediction on a single example is called the loss function. Usually, this function has a fixed number of parameters, and the total loss on a set is defined to be the sum of the loss on each element of the set, using the same parameters for each element. Also in general, a learning algorithm has two parts: first the definition of a loss function with parameters, and second a specific algorithm to find values for the parameters (also called settings, weights, or coefficients) that minimize the total loss given a training set. For the same loss function, there may be more than one training algorithm. Also, some algorithms may only find a local optimum of the loss function.

The optimal coefficient values $\hat{b}$ are not defined uniquely if the number n of training examples is less than the number p of features. Even if $n > p$ is true, the optimal coefficients can have multiple equivalent values, if some features are themselves related linearly. For an intuitive example, suppose features 1 and 2 are

height and weight respectively. Suppose that $x_2 = 120+5(x_1-60) = -180+5x_1$ approximately, where the units of measurement are pounds and inches. Then the same model can be written in many different ways:

- $y = b_0 + b_1x_1 + b_2x_2$
- $y = b_0 + b_1x_1 + b_2(-180 + 5x_1) = [b_0 - 180b_2] + [b_1(1 + 5b_2)]x_1 + 0x_2$

and more. In the extreme, suppose $x_1 = x_2$. Then all models $y = b_0+b_1x_1+b_2x_2$ are equivalent for which $b_1 + b_2$ equals a constant.

When two or more features are approximately related linearly, then the true values of the coefficients of those features are not well determined. The coefficients obtained by training will be strongly influenced by randomness in the training data. Regularization is a way to reduce the influence of this type of randomness. Consider all models $y = b_0 + b_1x_1 + b_2x_2$ for which $b_1 + b_2 = c$. Among these models, there is a unique one that minimizes the function $b_1^2 + b_2^2$. This model has $b_1 = b_2 = c/2$. We can obtain it by setting the objective function for training to be the sum of squared errors (SSE) plus a function that penalizes large values of the coefficients. A simple penalty function of this type is $\sum_{j=1}^{p} b_j^2$. A parameter λ can control the relative importance of the two objectives, namely SSE and penalty:

$$\hat{b} = \text{argmin}_b \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} b_j^2.$$

If $\lambda = 0$ then one gets the standard least-squares linear regression solution. As λ gets larger, the penalty on large coefficients gets stronger, and the typical values of coefficients get smaller. The parameter λ is often called the strength of regularization.

The penalty function $\sum_{j=1}^{p} b_j^2$ is only sensible if the typical magnitude is similar for the values of each feature. This an important motivation for data normalization. Note that in the formula $\sum_{j=1}^{p} b_j^2$ the sum excludes the intercept coefficient b_0. One reason for doing this is that the target y values are typically not normalized.

CSE 250B Quiz for Thursday January 7, 2010

Your name:

While taking this quiz you may use only your own handwritten notes, and printed copies of CSE 250B lecture notes.

The LAESA algorithm is as follows, given a test point x:

compute $d(b, x)$ for all $b \in B$
for all $t \in T$
    compute lower bound $g(t, x) = \max_b |d(t, b) - d(b, x)|$
initialize $y = \text{argmin}_b\ d(b, x)$
for all r in T sorted by increasing $g(r, x)$
    if $g(r, x) \geq d(y, x)$ then return y and finish
    else compute $d(r, x)$
        if $d(r, x) < d(y, x)$ then set $y = r$

The algorithm uses a fixed set of basis points B. Distances are computed just once between all training points and all points in the set B, in a preprocessing stage. We did not specify in class how to choose the points in B.

Question [4 points]: Is it better to select points in B that are (a) far from each other, or (b) close to each other? Justify your answer in one or two sentences.

CSE 250B Quiz for Thursday January 8, 2011

Your name:

While taking this quiz you may use only your own handwritten notes, and printed copies of CSE 250B lecture notes.

The standard Euclidean distance function is

$$d(x, y) = (\sum_{i=1}^{m} (x_i - y_i)^2)^{1/2}.$$

In a nearest-neighbor algorithm, instead of using this distance function, one could use a scalar product similarity function:

$$s(x, y) = \sum_{i=1}^{m} x_i y_i$$

Explain two reasons why the distance function is preferable. Answer in a few English sentences. If you can, use a mathematical identity as part of your answer.

Sample answer by Aditya Menon: There are several reasons why $d(x, y)$ is preferable. First, $s(x, 0) = 0$ for every x, so the 0 vector is equally close to all points, which is not sensible. Second, $s(x, x)$ is not the minimum/maximum of the similarity function, so even if a query point is identical to a training example, it may not be returned as the nearest neighbor. Third, $s(x, y)$ does not satisfy the triangle inequality, so it is not a sensible notion of distance.

Additional comments: The identity relating $d(x, y)$ and $s(x, y)$ is

$$d^2(x, y) = x \cdot x - 2x \cdot y + y \cdot y.$$

If all points have the same magnitude $||x|| = \sqrt{x \cdot x}$ then distance and negative similarity will give the same ranking of neighbors. With x fixed, the nearest neighbors y by distance have largest values of

$$2x \cdot y - y \cdot y$$

while the nearest by similarity have largest values of $2x \cdot y$. Everything else being equal, distance penalizes vectors y with large magnitude. This is reasonable, for

example because it is intuitive to say that the vector $y = 0$ is more similar to x than a vector that is orthogonal to x and nonzero.

Grading guide: Scores are integers between 0 and 4. The reasons why points can be lost include the following. This list is more or less in order of importance from most serious to least serious: statements that are incorrect, major mathematical errors, no good justification given, reasons that are valid but less important than other reasons that are not stated, statements that are true but not relevant, minor mathematical errors, sentences that are so complicated they are hard to understand, and errors in English that make writing hard to understand.

Assignment 1

This assignment is due at the start of class on Thursday January 20, 2011. You should work in a group of three students for this project. If enrollment declines, then we will have groups of two for later projects.

The goal of this project is to learn to recognize handwritten digits. Specifically, you will investigate various nearest-neighbor classifiers. The input to each classifier is a 16 by 16 array of pixels. The output of the classifier is one of the ten digits from 0 and 9. There are 7291 examples to be used for training, and 2007 examples for testing. Each pixel is a gray-scale intensity between -1 and +1. Note that although each digit is a two-dimensional array, for the project you should treat it as a one-dimensional vector of length $16^2 = 256$.

This dataset is called the USPS dataset, and results on it have been published in many papers. However, it is always important to check whether different papers use the exact same version of a dataset. It happens often that different papers use the same name for alternative versions of a dataset. The specific data to use now are available at `http://www-stat-class.stanford.edu/~tibs/ElemStatLearn/data.html`. Do read the info file carefully to understand this particular version of the data.

Your assignment is to compare different variants of the k nearest-neighbor classification method (kNN). Specifically, use Matlab or R to implement and do experiments with (i) kNN using Euclidean distance, and (ii) kNN using a learned similarity function as explained in class on January 6. For each version of kNN, select how many nearest neighbors k to use in a sensible way. Design a sensible tie-breaking heuristic, and describe it clearly. When learning a similarity function using the approach explained in class, there are many algorithmic design options. Explain your choices, and reasons for them, in your report.

Do experiments with the multiclass task of distinguishing all ten digits. Think rigorously about designing experiments that are conceptually sound. For final accuracy measurements, use the standard test set of 2007 examples. Compare your performance with published results using other methods from at least one published paper, for example [Shivaswamy and Jebara, 2008]. Do not overfit the test set!

For this project and for later ones, the only deliverable is a well-written joint report for each team. As is the case for a research paper, your job is to inform and convince the reader fully. Please bring a printed copy to the start of class on January 20.

To quote Sanjoy Dasgupta, "Discuss your results in precise and lucid prose. Content is king, but looks matter too!" Your report should be self-contained and complete, but concise. You should have separate sections for at least the following: (i) introduction, (ii) design of algorithms, (iii) design of experiments, (iv) results of experiments, and (v) findings and lessons learned.

The report should be typeset with LaTeX and have appropriate equations, tables, figures, citations, and references. Explain in reproducible detail your algorithms, especially

the heuristics that you invent, and the design of your experiments. These explanations should be separate from actual experimental results. Some issues to discuss briefly in the report include whether the training and test sets follow the same probability distribution, and whether overfitting is a problem. Analyze the time and space complexity of your algorithms, and provide brief timing results. Explain briefly the implementation choices that allow your code for the algorithms to be fast.

Optional extra work

Optionally, if you have time and enthusiasm, implement the LAESA algorithm. Design your own simple heuristic for choosing the basis points. Do informal experiments to select how many basis points to use. An important question is whether LAESA, or any other method, can be implemented so that it actually is faster than "brute force" computation of all distances. Even if LAESA can be implemented to be faster in some languages, it may be slower in Matlab because Matlab performs large-scale matrix arithmetic very fast, but updating data structures and performing logical tests in Matlab tends to be slow. If you can make LAESA faster than brute-force search in Matlab, explain how in your report.

Matlab notes

The following is the fastest Matlab code that I know for calculating Euclidean distances between a test point and all points in a training set. It is fast because it uses the identity $(a-b)^2 = a^2 - 2ab + b^2$ to avoid copying data.

```
function distances = calcdist(data,point)
%  input: data matrix, single point (points are row vectors)
% output: column vector of distances
distances = sum(data.^2, 2) - 2*data*point' + point*point';
distances = sqrt(distances);
```

Use this code as a starting point for your work. If you can make it notably faster, please share your code on the 250B message board.

For other fast Matlab subroutines, use the Lightspeed toolbox published by Minka at `http://research.microsoft.com/en-us/um/people/minka/software/lightspeed/`.

References

[Aggarwal et al., 2001] Aggarwal, C. C., Hinneburg, A., and Keim, D. A. (2001). On the surprising behavior of distance metrics in high dimensional spaces. In den Bussche, J. V. and Vianu, V., editors, *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings*, volume 1973 of *Lecture Notes in Computer Science*, pages 420–434. Springer.

[Devroye et al., 1994] Devroye, L., Gyorfi, L., Krzyzak, A., and Lugosi, G. (1994). On the strong universal consistency of nearest neighbor regression function estimates. *Annals of Statistics*, 22:1371–1385.

[Kibriya and Frank, 2007] Kibriya, A. M. and Frank, E. (2007). An empirical comparison of exact nearest neighbour algorithms. In *Proceedings of the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'07)*, volume 4702 of *Lecture Notes in Computer Science*, pages 140–151. Springer.

[Ripley, 1996] Ripley, B. (1996). *Pattern Recognition and Neural Networks*. Cambridge University Press.

[Shivaswamy and Jebara, 2008] Shivaswamy, P. K. and Jebara, T. (2008). Relative margin machines. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *NIPS*, pages 1481–1488. MIT Press.