CHAPTER 6

Similarity, Neighbors, and Clusters

Fundamental concepts: *Calculating similarity of objects described by data; Using similarity for prediction; Clustering as similarity-based segmentation.*

Exemplary techniques: *Searching for similar entities; Nearest neighbor methods; Clustering methods; Distance metrics for calculating similarity.*

Similarity underlies many data science methods and solutions to business problems. If two things (people, companies, products) are similar in some ways they often share other characteristics as well. Data mining procedures often are based on grouping things by similarity or searching for the "right" sort of similarity. We saw this implicitly in previous chapters where modeling procedures create boundaries for grouping instances together that have similar values for their target variables. In this chapter we will look at similarity directly, and show how it applies to a variety of different tasks. We include sections with some technical details, in order that the more mathematical reader can understand similarity in more depth; these sections can be skipped.

Different sorts of business tasks involve reasoning from similar examples:

- We may want to *retrieve* similar things directly. For example, IBM wants to find companies that are similar to their best business customers, in order to have the sales staff look at them as prospects. Hewlett-Packard maintains many high-performance servers for clients; this maintenance is aided by a tool that, given a server configuration, retrieves information on other similarly configured servers. Advertisers often want to serve online ads to consumers who are similar to their current good customers.
- Similarity can be used for doing *classification* and *regression*. Since we now know a good bit about classification, we will illustrate the use of similarity with a classification example below.
- We may want to group similar items together into *clusters*, for example to see whether our customer base contains groups of similar customers and what these

groups have in common. Previously we discussed supervised segmentation; this is unsupervised segmetation. After discussing the use of similarity for classification, we will discuss its use for clustering.

- Modern retailers such as Amazon and Netflix use similarity to provide *recommendations* of similar products or from similar people. Whenever you see statements like "People who like X also like Y" or "Customers with your browsing history have also looked at …" similarity is being applied. In Chapter 12, we will discuss how a customer can be similar to a movie, if the two are described by the same "taste dimensions." In this case, to make recommendations we can find the movies that are most similar to the customer (and which the customer has not already seen).
- Reasoning from similar cases of course extends beyond business applications; it is natural to fields such as medicine and law. A doctor may reason about a new difficult case by recalling a similar case (either treated personally or documented in a journal) and its diagnosis. A lawyer often argues cases by citing legal precedents, which are similar historical cases whose dispositions were previously judged and entered into the legal casebook. The field of Artificial Intelligence has a long history of building systems to help doctors and lawyers with such case-based reasoning. Similarity judgments are a key component.

In order to discuss these applications further, we need to take a minute to formalize similarity and its cousin, distance.

Similarity and Distance

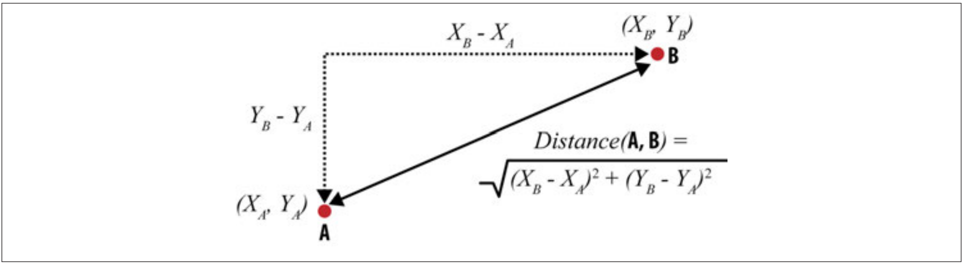
Once an object can be represented as data, we can begin to talk more precisely about the similarity between objects, or alternatively the distance between objects. For example, let's consider the data representation we have used throughout the book so far: represent each object as a feature vector. Then, the closer two objects are in the space defined by the features, the more similar they are.

Recall that when we build and apply predictive models, the goal is to determine the value of a target characteristic. In doing so, we've used the implicit similarity of objects already. "Visualizing Segmentations" on page 67 discussed the geometric interpretation of some classification models and "Classification via Mathematical Functions" on page 83 discussed how two different model types divide up an instance space into regions based on closeness of instances with similar class labels. Many methods in data science may be seen in this light: as methods for organizing the space of data instances (representations of important objects) so that instances near each other are treated similarly for some purpose. Both classification trees and linear classifiers establish boundaries between regions of differing classifications. They have in common the view that instances sharing a common region in space should be similar; what differs between the methods is how the regions are represented and discovered.

So why not reason about the similarity or distance between objects directly? To do so, we need a basic method for measuring similarity or distance. What does it mean that two companies or two consumers are similar? Let's examine this carefully. Consider two instances from our simplified credit application domain:

Attribute	Person A	Person B
Age	23	40
Years at current address	2	10
Residential status (1=Owner, 2=Renter, 3=Other)	2	1

These data items have multiple attributes, and there's no single best method for reducing them to a single similarity or distance measurement. There are many different ways to measure the similarity or distance between Person A and Person B. A good place to begin is with measurements of distance from basic geometry.



$X_B - X_A$
(X_B, Y_B)
B
$Y_B - Y_A$
$Distance(\mathbf{A}, \mathbf{B}) = \sqrt{(X_B - X_A)^2 + (Y_B - Y_A)^2}$
(X_A, Y_A)
A

Figure 6-1. Euclidean distance.

Recall from our prior discussions of the geometric interpretation that if we have two (numeric) features, then each object is a point in a two-dimensional space. Figure 6-1 shows two data items, A and B, located on a two-dimensional plane. Object A is at coordinates (x_A, y_A) and B is at (x_B, y_B). At the risk of too much repetition, note that these coordinates are just the values of the two features of the objects. We can draw a right triangle between the two objects, as shown, whose base is the difference in the x's: $(x_A - x_B)$ and whose height is the difference in the y's: $(y_A - y_B)$. The Pythagorean theorem tells us that the distance between A and B is given by the length of the hypotenuse, and is equal to the square root of the summed squares of the lengths of the other two sides of the triangle, which in this case is $\sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$. Essentially, we can compute the overall distance by computing the distances of the individual dimensions—the individual features in our setting. This is called the *Euclidean distance* [1] between two points, and it's probably the most common geometric distance measure.

1. After Euclid, the 4th century B.C. Greek mathematician known as the Father of Geometry.

Euclidean distance is not limited to two dimensions. If A and B were objects described by three features, they could be represented by points in three-dimensional space and their positions would then be represented as (x_A, y_A, z_A) and (x_B, y_B, z_B). The distance between A and B would then include the term $(z_A - z_B)^2$. We can add arbitrarily many features, each a new dimension. When an object is described by n features, n dimensions $(d_1, d_2, \ldots, d_n)$, the general equation for Euclidean distance in n dimensions is shown in Equation 6-1:

Equation 6-1. General Euclidean distance

$$\sqrt{(d_{1,A} - d_{1,B})^2 + (d_{2,A} - d_{2,B})^2 + \ \ldots + (d_{n,A} - d_{n,B})^2}$$

We now have a metric for measuring the distance between any two objects described by vectors of numeric features—a simple formula based on the distances of the objects' individual features. Recalling persons A and B, above, their Euclidean distance is:

$$\begin{aligned} d(A,\ B) &= \sqrt{(23 - 40)^2 + (2 - 10)^2 + (2 - 1)^2} \\ &\approx 18.8 \end{aligned}$$

So the distance between these examples is about 19. This distance is just a number—it has no units, and no meaningful interpretation. It is only really useful for comparing the similarity of one pair of instances to that of another pair. It turns out that comparing similarities is extremely useful.

Nearest-Neighbor Reasoning

Now that we have a way to measure distance, we can use it for many different data-analysis tasks. Recalling examples from the beginning of the chapter, we could use this measure to find the companies most similar to our best corporate customers, or the online consumers most similar to our best retail customers. Once we have found these, we can take whatever action is appropriate in the business context. For corporate customers, IBM does this to help direct its sales force. Online advertisers do this to target ads. These most-similar instances are called *nearest neighbors*.

Example: Whiskey Analytics

Let's talk about a fresh example. One of us (Foster) likes single malt Scotch whiskey. If you've had more than one or two, you realize that there is a lot of variation among the hundreds of different single malts. When Foster finds a single malt he really likes, he wants to find other similar ones—both because he likes to explore the "space" of single malts, but also because any given liquor store or restaurant only has a limited selection.

He wants to be able to pick one he'll really like. For example, the other evening a dining companion recommended trying the single malt "Bunnahabhain."[2] It was unusual and very good. Out of all the many single malts, how could Foster find other ones like that?

Let's take a data science approach. Recall from Chapter 2 that we first should think about the exact question we would like to answer, and what are the appropriate data to answer it. How can we describe single malt Scotch whiskeys as feature vectors, in such a way that we think similar whiskeys will have similar taste? This is exactly the project undertaken by François-Joseph Lapointe and Pierre Legendre of the University of Montréal (Lapointe & Legendre, 1994). They were interested in several classification and organizational questions about Scotch whiskeys. We'll adopt some of their approach here.

It turns out that tasting notes are published for many whiskeys. For example, Michael Jackson is a well-known whiskey and beer connoisseur who has written *Michael Jackson's Malt Whisky Companion: A Connoisseur's Guide to the Malt Whiskies of Scotland* (Jackson, 1989), which describes 109 different single malt Scotches of Scotland. The descriptions are in the form of tasting notes on each Scotch, such as: "*Appetizing aroma of peat smoke, almost incense-like, heather honey with a fruity softness.*"

As data scientists, we are making progress. We have found a potentially useful source of data. However, we do not yet have whiskeys described by feature vectors, only by tasting notes. We need to press on with our data formulation. Following Lapointe and Legendre (1994), let's create some numeric features that, for any whiskey, will summarize the information in the tasting notes. Define five general whiskey attributes, each with many possible values:

1.	**Color:** *yellow, very pale, pale, pale gold, gold, old gold, full gold, amber,* etc.	(14 values)
2.	**Nose:** *aromatic, peaty, sweet, light, fresh, dry, grassy,* etc.	(12 values)
3.	**Body:** *soft, medium, full, round, smooth, light, firm, oily.*	(8 values)
4.	**Palate:** *full, dry, sherry, big, fruity, grassy, smoky, salty,* etc.	(15 values)
5.	**Finish:** *full, dry, warm, light, smooth, clean, fruity, grassy, smoky,* etc.	(19 values)

It is important to note that these category values are *not* mutually exclusive (e.g., Aberlour's palate is described as medium, full, soft, round and smooth). In general, any of the values can co-occur (though some of them, like Color being both light and smoky, never do) but because they can co-occur, each value of each variable was coded as a separate feature by Lapointe and Legendre. Consequently there are 68 binary features of each whiskey.

2. No, he can't pronounce it properly either.

Foster likes Bunnahabhain, so we can use Lapointe and Legendre's representation of whiskeys with Euclidean distance to find similar ones for him. For reference, here is their description of Bunnahabhain:

- *Color:* gold
- *Nose:* fresh and sea
- *Body:* firm, medium, and light
- *Palate:* sweet, fruity, and clean
- *Finish:* full

Here is Bunnahabhain's description and the five single-malt Scotches most similar to Bunnahabhain, by increasing distance:

Whiskey	Distance	Descriptors
Bunnahabhain	—	*gold; firm,med,light; sweet,fruit,clean; fresh,sea; full*
Glenglassaugh	0.643	gold; firm,light,smooth; sweet,grass; fresh,grass
Tullibardine	0.647	gold; firm,med,smooth; sweet,fruit,full,grass,clean; sweet; big,arome,sweet
Ardberg	0.667	sherry; firm,med,full,light; sweet; dry,peat,sea;salt
Bruichladdich	0.667	pale; firm,light,smooth; dry,sweet,smoke,clean; light; full
Glenmorangie	0.667	p.gold; med,oily,light; sweet,grass,spice; sweet,spicy,grass,sea,fresh; full,long

Using this list we could find a Scotch similar to Bunnahabhain. At any particular shop we might have to go down the list a bit to find one they stock, but since the Scotches are ordered by similarity we can easily find the most similar Scotch (and also have a vague idea as to how similar the closest available Scotch is as compared to the alternatives that are not available).

This is an example of the direct application of similarity to solve a problem. Once we understand this fundamental notion, we have a powerful conceptual tool for approaching a variety of problems, such as those laid out above (finding similar companies, similar consumers, etc.). As we see in the whiskey example, the data scientist often still has work to do to actually define the data so that the similarity will be with respect to a useful set of characteristics. Later we will present some other notions of similarity and distance. Now, let's move on to another very common use of similarity in data science.

Nearest Neighbors for Predictive Modeling

We also can use the idea of nearest neighbors to do predictive modeling in a different way. Take a minute to recall everything you now know about predictive modeling from prior chapters. To use similarity for predictive modeling, the basic procedure is beautifully simple: given a new example whose target variable we want to predict, we scan through all the training examples and choose several that are the most similar to the

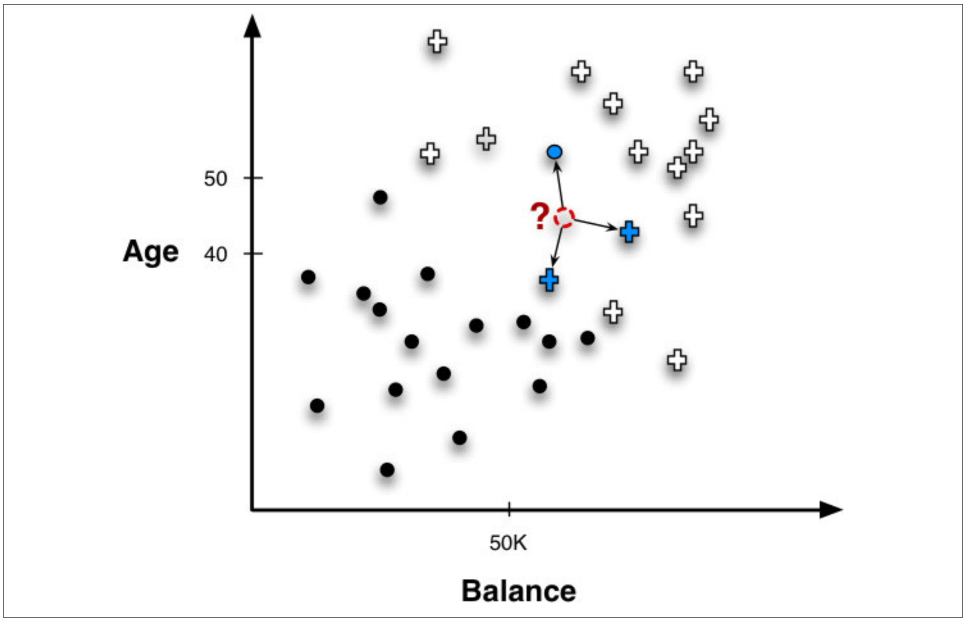


Figure 6-2. Nearest neighbor classification. The point to be classified, labeled with a question mark, would be classified + because the majority of its nearest (three) neighbors are +.

new example. Then we predict the new example's target value, based on the nearest neighbors' (known) target values. How to do that last step needs to be defined; for now, let's just say that we have some *combining function* (like voting or averaging) operating on the neighbors' known target values. The combining function will give us a prediction.

Classification

Since we have focused a great deal on classification tasks so far in the book, let's begin by seeing how neighbors can be used to classify a new instance in a super-simple setting. Figure 6-2 shows a new example whose label we want to predict, indicated by a "?." Following the basic procedure introduced above, the nearest neighbors (in this example, three of them) are retrieved and their known target variables (classes) are consulted. In this this case, two examples are positive and one is negative. What should be our combining function? A simple combining function in this case would be majority vote, so the predicted class would be positive.

Adding just a little more complexity, consider a credit card marketing problem. The goal is to predict whether a new customer will respond to a credit card offer based on how other, similar customers have responded. The data (still oversimplified of course) are shown in Table 6-1.

Table 6-1. Nearest neighbor example: Will David respond or not?

Customer	Age	Income (1000s)	Cards	Response (target)	Distance from David
David	*37*	*50*	*2*	*?*	0
John	35	35	3	Yes	$\sqrt{(35-37)^2+(35-50)^2+(3-2)^2}=15.16$
Rachael	22	50	2	No	$\sqrt{(22-37)^2+(50-50)^2+(2-2)^2}=15$
Ruth	63	200	1	No	$\sqrt{(63-37)^2+(200-50)^2+(1-2)^2}=152.23$
Jefferson	59	170	1	No	$\sqrt{(59-37)^2+(170-50)^2+(1-2)^2}=122$
Norah	25	40	4	Yes	$\sqrt{(25-37)^2+(40-50)^2+(4-2)^2}=15.74$

In this example data, there are five existing customers we previously have contacted with a credit card offer. For each of them we have their name, age, income, the number of cards they already have, and whether they responded to the offer. For a new person, David, we want to predict whether he will respond to the offer or not.

The last column in Table 6-1 shows a distance calculation, using Equation 6-1, of how far each instance is from David. Three customers (John, Rachael, and Norah) are fairly similar to David, with a distance of about 15. The other two customers (Ruth and Jefferson) are much farther away. Therefore, David's three nearest neighbors are Rachael, then John, then Norah. Their responses are No, Yes, and Yes, respectively. If we take a majority vote of these values, we predict Yes (David will respond). This touches upon some important issues with nearest-neighbor methods: how many neighbors should we use? Should they have equal weights in the combining function? We discuss these later in the chapter.

Probability Estimation

We've made the point that it's usually important not just to classify a new example but to estimate its probability—to assign a score to it, because a score gives more information than just a Yes/No decision. Nearest neighbor classification can be used to do this fairly easily. Consider again the classification task of deciding whether David will be a responder or not. His nearest neighbors (Rachael, John, and Norah) have classes of No, Yes, and Yes, respectively. If we score for the Yes class, so that Yes=1 and No=0, we can average these into a score of *2/3* for David. If we were to do this in practice, we might want to use more than just three nearest neighbors to compute the probability estimates (and recall the discussion of estimating probabilities from small samples in "Probability Estimation" on page 71).

Regression

Once we can retrieve nearest neighbors, we can use them for any predictive mining task by combining them in different ways. We just saw how to do classification by taking a majority vote of a target. We can do regression in a similar way.

Assume we had the same dataset as in Table 6-1, but this time we want to predict David's Income. We won't redo the distance calculation, but assume that David's three nearest neighbors were again Rachael, John, and Norah. Their respective incomes are 50, 35, and 40 (in thousands). We then use these values to generate a prediction for David's income. We could use the average (about 42) or the median (40).

It is important to note that in retrieving neighbors we do not use the target variable because we're trying to predict it. Thus Income would not enter into the distance calculation as it does in Table 6-1. However, we're free to use any other variables whose values are available to determine distance.

How Many Neighbors and How Much Influence?

In the course of explaining how classification, regression, and scoring may be done, we have used an example with only three neighbors. Several questions may have occurred to you. First, why *three* neighbors, instead of just one, or five, or one hundred? Second, should we treat all neighbors the same? Though all are called "nearest" neighbors, some are nearer than others, and shouldn't this influence how they're used?

There is no simple answer to how many neighbors should be used. Odd numbers are convenient for breaking ties for majority vote classification with two-class problems. Nearest neighbor algorithms are often referred to by the shorthand k-NN, where the k refers to the number of neighbors used, such as 3-NN.

In general, the greater k is the more the estimates are smoothed out among neighbors. If you have understood everything so far, with a little thought you should realize that if we increase k to the maximum possible (so that $k = n$) the entire dataset would be used for every prediction. Elegantly, this simply predicts the average over the entire dataset for any example. For classification, this would predict the majority class in the entire dataset; for regression, the average of all the target values; for class probability estimation, the "base rate" probability (see Note: Base rate in "Holdout Data and Fitting Graphs" on page 113).

Even if we're confident about the number of neighbor examples we should use, we may realize that neighbors have different similarity to the example we're trying to predict. Shouldn't this influence how they're used?

For classification we started with a simple strategy of *majority* voting, retrieving an odd number of neighbors to break ties. However, this ignores an important piece of information: how close each neighbor is to the instance. For example, consider what would happen if we used $k = 4$ neighbors to classify David. We would retrieve the responses (Yes, No, Yes, No), causing the responses to be evenly mixed. But the first three are very close to David (distance ≈ 15) while the fourth is much further away (distance ≈ 122).

Intuitively, this fourth instance shouldn't contribute as much to the vote as the first three. To incorporate this concern, nearest-neighbor methods often use *weighted voting* or *similarity moderated voting* such that each neighbor's contribution is scaled by its similarity.

Consider again the data in Table 6-1, involving predicting whether David will respond to a credit card offer. We showed that if we predict David's class by majority vote it depends greatly on the number of neighbors we choose. Let's redo the calculations, this time using *all* neighbors but scaling each by its similarity to David, using as the scaling weight the reciprocal of the square of the distance. Here are the neighbors ordered by their distance from David:

Name	Distance	Similarity weight	Contribution	Class
Rachael	15.0	0.004444	0.344	No
John	15.2	0.004348	0.336	Yes
Norah	15.7	0.004032	0.312	Yes
Jefferson	122.0	0.000067	0.005	No
Ruth	152.2	0.000043	0.003	No

The Contribution column is the amount that each neighbor contributes to the final calculation of the target probability prediction (the contributions are proportional to the weights, but adding up to one). We see that distances greatly effect contributions: Rachael, John and Norah are most similar to David and effectively determine our prediction of his response, while Jefferson and Ruth are so far away that they contribute virtually nothing. Summing the contributions for the positive and negative classes, the final probability estimates for David are 0.65 for Yes and 0.35 for No.

This concept generalizes to other sorts of prediction tasks, for example regression and class probability estimation. Generally, we can think of the procedure as weighted *scoring*. Weighted scoring has a nice consequence in that it reduces the importance of deciding how many neighbors to use. Because the contribution of each neighbor is moderated by its distance, the influence of neighbors naturally drops off the farther they are from the instance. Consequently, when using weighted scoring the exact value of k is much less critical than with majority voting or unweighted averaging. Some methods avoiding committing to a k by retrieving a very large number of instances (e.g., all instances, $k = n$) and depend upon distance weighting to moderate the influences.

Sidebar: Many names for nearest-neighbor reasoning

As with many things in data mining, different terms exist for nearest-neighbor classifiers, in part because similar ideas were pursued independently. Nearest-neighbor classifiers were established long ago in statistics and pattern recognition (Cover & Hart, 1967). The idea of classifying new instances directly by consulting a database (a "memory") of instances has been termed *instance-based learning* (Aha, Kibler, & Albert, 1991) and *memory-based learning* (Lin & Vitter, 1994). Because no model is built during "training" and most effort is deferred until instances are retrieved, this general idea is known as *lazy learning* (Aha, 1997).

A related technique in artificial intelligence is *Case-Based Reasoning* (Kolodner, 1993; Aamodt & Plaza, 1994), abbreviated CBR. Past cases are commonly used by doctors and lawyers to reason about new cases, so case-based reasoning has a well-established history in these fields.

However, there are also significant differences between case-based reasoning and nearest-neighbor methods. Cases in CBR are typically not simple feature vector instances but instead are very detailed summaries of an episode, including items such as a patient's symptoms, medical history, diagnosis, treatment, and outcome; or the details of a legal case including plaintiff and defendant arguments, precedents cited, and judgement. Because cases are so detailed, in CBR they are used not just to provide a class label but to provide diagnostic and planning information that can be used to deal with the case after it is retrieved. Adapting historical cases to be used in a new situation is usually a complex process that requires significant effort.

Geometric Interpretation, Overfitting, and Complexity Control

As with other models we've seen, it is instructive to visualize the classification regions created by a nearest-neighbor method. Although no explicit boundary is created, there are implicit regions created by instance neighborhoods. These regions can be calculated by systematically probing points in the instance space, determining each point's classification, and constructing the boundary where classifications change.

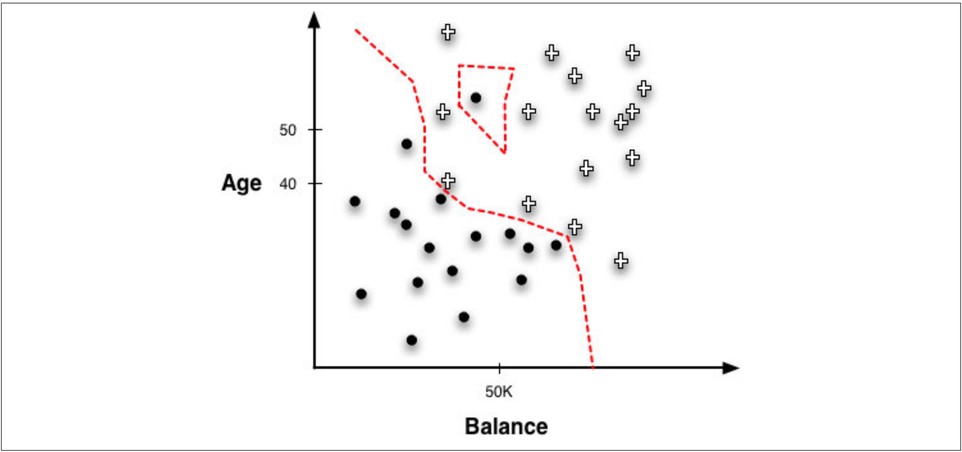


Figure 6-3. Boundaries created by a 1-NN classifier.

Figure 6-3 illustrates such a region created by a 1-NN classifier around the instances of our "Write-off" domain. Compare this with the classification tree regions from Figure 3-15 and the regions created by the linear boundary in Figure 4-3.

Notice that the boundaries are not lines, nor are they even any recognizable geometric shape; they are erratic and follow the frontiers between training instances of different classes. The nearest-neighbor classifier follows very specific boundaries around the training instances. Note also the one negative instance isolated inside the positive instances creates a "negative island" around itself. This point might be considered noise or an outlier, and another model type might smooth over it.

Some of this sensitivity to outliers is due to the use of a 1-NN classifier, which retrieves only single instances, and so has a more erratic boundary than one that averages multiple neighbors. We will return to that in a minute. More generally, irregular concept boundaries are characteristic of all nearest-neighbor classifiers, because they do not impose any particular geometric form on the classifier. Instead, they form boundaries in instance space tailored to the specific data used for training.

This should recall our discussions of overfitting and complexity control from Chapter 5. If you're thinking that 1-NN must overfit very strongly, then you are correct. In fact, think about what would happen if you evaluated a 1-NN classifier on the training data. When classifying each training data point, any reasonable distance metric would lead to the retrieval of that training point itself as its own nearest neighbor! Then its own value for the target variable would be used to predict itself, and voilà, perfect classification. The same goes for regression. The 1-NN memorizes the training data. It does a little better than our strawman lookup table from the beginning of Chapter 5, though. Since the lookup table did not have any notion of similarity, it simply predicted perfectly for exact training examples, and gave some default prediction for all others. The 1-NN classifier predicts perfectly for training examples, but it also can make an often reasonable prediction on other examples: it uses the most similar training example.

Thus, in terms of overfitting and its avoidance, the k in a k-NN classifier is a complexity parameter. At one extreme, we can set $k = n$ and we do not allow much complexity at all in our model. As described previously, the n-NN model (ignoring similarity weighting) simply predicts the average value in the dataset for each case. At the other extreme, we can set $k = 1$, and we will get an extremely complex model, which places complicated boundaries such that every training example will be in a region labeled by its own class.

Now let's return to an earlier question: how should one choose k? We can use the same procedure discussed in "A General Method for Avoiding Overfitting" on page 134 for setting other complexity parameters: we can conduct cross-validation or other nested holdout testing on the training set, for a variety of different values of k, searching for one that gives the best performance on the training data. Then when we have chosen a value of k, we build a k-NN model from the entire training set. As discussed in detail in Chapter 5, since this procedure only uses the training data, we can still evaluate it on the test data and get an unbiased estimate of its generalization performance. Data mining tools usually have the ability to do such nested cross-validation to set *k* automatically.

Figure 6-4 and Figure 6-5 show different boundaries created by nearest-neighbor classifiers. Here a simple three-class problem is classified using different numbers of neighbors. In Figure 6-4, only a single neighbor is used, and the boundaries are erratic and very specific to the training examples in the dataset. In Figure 6-5, 30 nearest neighbors are averaged to form a classification. The boundaries are obviously different from Figure 6-4 and are much less jagged. Note, however, that in neither case are the boundaries smooth curves or regular piecewise geometric regions that we would expect to see with a linear model or a tree-structured model. The boundaries for *k*-NN are more strongly defined by the data.

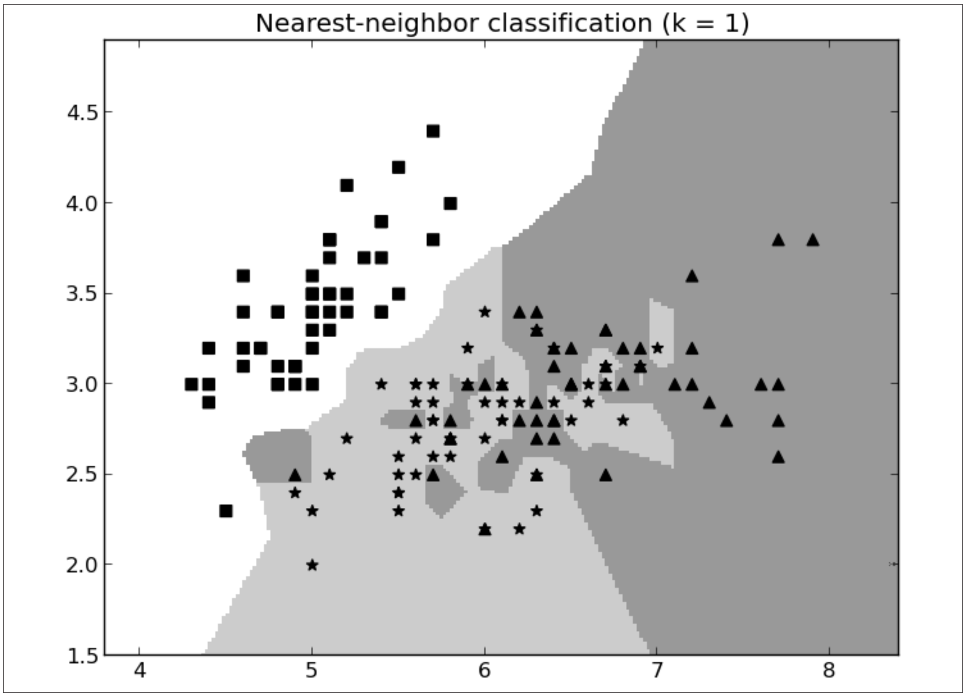


Figure 6-4. Classification boundaries created on a three-class problem created by 1-NN (single nearest neighbor).

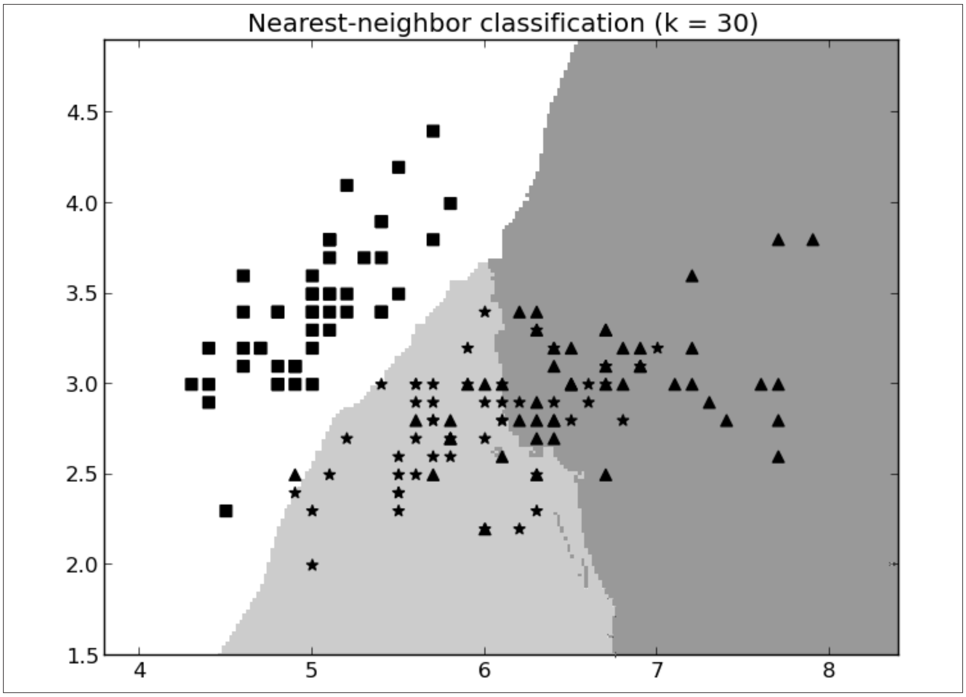


Figure 6-5. Classification boundaries created on a three-class problem created by 30-NN (averaging 30 nearest neighbors).

Issues with Nearest-Neighbor Methods

Before concluding a discussion of nearest-neighbor methods as predictive models, we should mention several issues regarding their use. These often come into play in real-world applications.

Intelligibility

Intelligibility of nearest-neighbor classifiers is a complex issue. As mentioned, in some fields such as medicine and law, reasoning about similar historical cases is a natural way of coming to a decision about a new case. In such fields, a nearest-neighbor method may be a good fit. In other areas, the lack of an explicit, interpretable model may pose a problem.

There are really two aspects to this issue of intelligibility: the justification of a specific *decision* and the intelligibility of an entire *model*.

With *k*-NN, it usually is easy to describe how a single instance is decided: the set of neighbors participating in the decision can be presented, along with their contributions. This was done for the example involving the prediction of whether David would respond, earlier in Table 6-1. Some careful phrasing and judicious presentation of nearest neighbors are useful. For example, Netflix uses a form of nearest-neighbor classification for their recommendations, and explains their movie recommendations with sentences like:

> "The movie *Billy Elliot* was recommended based on your interest in *Amadeus*, *The Constant Gardener* and *Little Miss Sunshine*"

Amazon presents recommendations with phrases like: "Customers with similar searches purchased…" and "Related to Items You've Viewed."

Whether such justifications are adequate depends on the application. An Amazon customer may be satisfied with such an explanation for why she got a recommendation. On the other hand, a mortgage applicant may not be satisfied with the explanation, "We declined your mortgage application because you remind us of the Smiths and the Mitchells, who both defaulted." Indeed, some legal regulations restrict the sorts of models that can be used for credit scoring to models for which very simple explanations can be given based on specific, important variables. For example, with a linear model, one may be able to say: "all else being equal, if your income had been $20,000 higher you would have been granted this particular mortgage."

It also is easy to explain how the entire nearest-neighbor model generally decides new cases. The idea of finding the most similar cases and looking at how they were classified, or what value they had, is intuitive to many.

What is difficult is to explain more deeply what "knowledge" has been mined from the data. If a stakeholder asks "What did your system learn from the data about my customers? On what basis does it make its decisions?" there may be no easy answer because there is no explicit model. Strictly speaking, the nearest-neighbor "model" consists of the entire case set (the database), the distance function, and the combining function. In two dimensions we can visualize this directly as we did in the prior figures. However, this is not possible when there are many dimensions. The knowledge embedded in this model is not usually understandable, so if model intelligibility and justification are critical, nearest-neighbor methods should be avoided.

Dimensionality and domain knowledge

Nearest-neighbor methods typically take into account all features when calculating the distance between two instances. "Heterogeneous Attributes" on page 157 below discusses one of the difficulties with attributes: numeric attributes may have vastly different ranges, and unless they are scaled appropriately the effect of one attribute with a wide range can swamp the effect of another with a much smaller range. But apart from this, there is a problem with having too many attributes, or many that are irrelevant to the similarity judgment.

For example, in the credit card offer domain, a customer database could contain much incidental information such as number of children, length of time at job, house size, median income, make and model of car, average education level, and so on. Conceivably some of these could be relevant to whether the customer would accept the credit card offer, but probably most would be irrelevant. Such problems are said to be high-dimensional—they suffer from the so-called *curse of dimensionality*—and this poses problems for nearest neighbor methods. Much of the reason and effects are quite technical,[3] but roughly, since all of the attributes (dimensions) contribute to the distance calculations, instance similarity can be confused and misled by the presence of too many irrelevant attributes.

There are several ways to fix the problem of many, possibly irrelevant attributes. One is *feature selection*, the judicious determination of features that should be included in the data mining model. Feature selection can be done manually by the data miner, using background knowledge as what attributes are relevant. This is one of the main ways in which a data mining team injects domain knowledge into the data mining process. As discussed in Chapter 3 and Chapter 5 there are also automated feature selection methods that can process the data and make judgments about which attributes give information about the target.

Another way of injecting domain knowledge into similarity calculations is to tune the similarity/distance function manually. We may know, for example, that the attribute *Number of Credit Cards* should have a strong influence on whether a customer accepts an offer for another one. A data scientist can tune the distance function by assigning different weights to the different attributes (e.g., giving a larger weight to *Number of Credit Cards*). Domain knowledge can be added not only because we believe we know what will be more predictive, but more generally because we know something about the similar entities we want to find. When looking for similar whiskeys, I may know that "peatiness" is important to my judging a single malt as tasting similar, so I could give *peaty* a higher weight in the similarity calculation. If another taste variable is unimportant, I could remove it or simply give it a low weight.

Computational efficiency

One benefit of nearest-neighbor methods is that training is very fast because it usually involves only storing the instances. No effort is expended in creating a model. The main computational cost of a nearest neighbor method is borne by the prediction/classification step, when the database must be queried to find nearest neighbors of a new instance. This can be very expensive, and the classification expense should be a consideration. Some applications require extremely fast predictions; for example, in online advertisement targeting, decisions may need to be made in a few tens of milliseconds. For such applications, a nearest neighbor method may be impractical.

3. For example, it turns out that for technical reasons, with large numbers of features, certain particular instances appear extremely frequently in other instances' sets of *k* nearest neighbors. These particular instances thereby have a very large influence on many classifications.

There are techniques for speeding up neighbor retrievals. Specialized data structures like kd-trees and hashing methods (Shakhnarovich, Darrell, & Indyk, 2005; Papadopoulos & Manolopoulos, 2005) are employed in some commerical database and data mining systems to make nearest neighbor queries more efficient. However, be aware that many small-scale and research data mining tools usually do not employ such techniques, and still rely on naive brute-force retrieval.

Some Important Technical Details Relating to Similarities and Neighbors

Heterogeneous Attributes

Up to this point we have been using Euclidean distance, showing that it was easy to calculate. If attributes are numeric and are directly comparable, the distance calculation is indeed straightforward. When examples contain complex, heterogeneous attributes things become more complicated. Consider another example in the same domain but with a few more attributes:

Attribute	Person A	Person B
Sex	Male	Female
Age	23	40
Years at current address	2	10
Residential status (1=Owner, 2=Renter, 3=Other)	2	1
Income	50,000	90,000

Several complications now arise. First, the equation for Euclidean distance is numeric, and Sex is a categorical (symbolic) attribute. It must be encoded numerically. For binary variables, a simple encoding like M=0, F=1 may be sufficient, but if there are multiple values for a categorical attribute this will not be good enough.

Also important, we have variables that, though numeric, have very different scales and ranges. Age might have a range from 18 to 100, while Income might have a range from \$10 to \$10,000,000. Without scaling, our distance metric would consider ten dollars of income difference to be as significant as ten years of age difference, and this is clearly wrong. For this reason nearest-neighbor-based systems often have variable-scaling front ends. They measure the ranges of variables and scale values accordingly, or they apportion values to a fixed number of bins. The general principle at work is that care must be taken that the similarity/distance computation is meaningful for the application.

* Other Distance Functions

For simplicity, up to this point we have used only a single metric, Euclidean distance. Here we include more details about distance functions and some alternatives.

It is important to note that the similarity measures presented here represent only a tiny fraction of all the similarity measures that have been used. These ones are particularly popular, but both the data scientist and the business analyst should keep in mind that it is important to use a meaningful similarity metric with respect to the business problem at hand. This section may be skipped without loss of continuity.

As noted previously, Euclidean distance is probably the most widely used distance metric in data science. It is general, intuitive and computationally very fast. Because it employs the *squares* of the distances along each individual dimension, it is sometimes called the *L2 norm* and sometimes represented by $|| \cdot ||_2$. Equation 6-2 shows how it looks formally.

Equation 6-2. Euclidean distance (L2 norm)

$$d_{\text{Euclidean}}(\mathbf{X}, \mathbf{Y}) = \| \mathbf{X} - \mathbf{Y} \|_2 = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots}$$

Though Euclidean distance is widely used, there are many other distance calculations. The *Dictionary of Distances* by Deza & Deza (Elsevier Science, 2006) lists several hundred, of which maybe a dozen or so are used regularly for mining data. The reason there are so many is that in a nearest-neighbor method the distance function is critical. It basically reduces a comparison of two (potentially complex) examples into a single number. The data types and specifics of the domain of application greatly influence how the differences in individual attributes should combine.

The *Manhattan distance* or *L1-norm* is the sum of the (*unsquared*) pairwise distances, as shown in Equation 6-3.

Equation 6-3. Manhattan distance (L1 norm)

$$d_{\text{Manhattan}}(\mathbf{X}, \mathbf{Y}) = \| \mathbf{X} - \mathbf{Y} \|_1 = | x_1 - y_1 | + | x_2 - y_2 | + \cdots$$

This simply sums the differences along the different dimensions between *X* and *Y*. It is called Manhattan (or taxicab) distance because it represents the total street distance you would have to travel in a place like midtown Manhattan (which is arranged in a grid) to get between two points—the total east-west distance traveled plus the total north-south distance traveled.

Researchers studying the whiskey analytics problem introduced above used another common distance metric.[4] Specifically, they used *Jaccard distance*. Jaccard distance treats the two objects as *sets* of characteristics. Thinking about the objects as sets allows one to think about the size of the union of all the characteristics of two objects X and Y, $|X \cup Y|$, and the size of the set of characteristics shared by the two objects (the intersection), $|X \cap Y|$. Given two objects, X and Y, the Jaccard distance is the proportion of all the characteristics (that either has) that are shared by the two. This is appropriate for problems where the possession of a common characteristic between two items is important, but the common *absence* of a characteristic is not. For example, in finding similar whiskeys it is significant if two whiskeys are both peaty, but it may not be significant that they are both not *salty*. In set notation, the Jaccard distance metric is shown in Equation 6-4.

Equation 6-4. Jaccard distance

$$d_{\text{Jaccard}}(X, Y) = 1 - \frac{|X \cap Y|}{|X \cup Y|}$$

Cosine distance is often used in text classification to measure the similarity of two documents. It is defined in Equation 6-5.

Equation 6-5. Cosine distance

$$d_{cosine}(\mathbf{X}, \mathbf{Y}) = 1 - \frac{\mathbf{X} \cdot \mathbf{Y}}{\|\mathbf{X}\|_2 \cdot \|\mathbf{Y}\|_2}$$

where $\|\cdot\|_2$ again represents the L2 norm, or Euclidean length, of each feature vector (for a vector this is simply the distance from the origin).

The information retrieval literature more commonly talks about *cosine similarity*, which is simply the fraction in Equation 6-5. Alternatively, it is 1 – cosine distance.

4. See Lapointe and Legendre (1994), Section 3 ("Classification of Pure Malt Scotch Whiskies"), for a detailed discussion of how they engineered their problem formulation. Available here.

In text classification, each word or token corresponds to a dimension, and the location of a document along each dimension is the number of occurrences of the word in that document. For example, suppose document A contains seven occurrences of the word *performance*, three occurrences of *transition*, and two occurrences of *monetary*. Document B contains two occurrences of *performance*, three occurrences of *transition*, and no occurrences of *monetary*. The two documents would be represented as vectors of counts of these three words: A = <7,3,2> and B = <2,3,0>. The cosine distance of the two documents is:

$$\begin{aligned} d_{\text{cosine}}(A, B) &= 1 - \frac{\langle 7, 3, 2\rangle \cdot \langle 2, 3, 0\rangle}{\| \langle 7, 3, 2\rangle \|_2 \cdot \| \langle 2, 3, 0\rangle \|_2} \\ &= 1 - \frac{7 \cdot 2 + 3 \cdot 3 + 2 \cdot 0}{\sqrt{49 + 9 + 4} \cdot \sqrt{4 + 9}} \\ &= 1 - \frac{23}{28.4} \approx 0.19 \end{aligned}$$

Cosine distance is particularly useful when you want to ignore differences in scale across instances—technically, when you want to ignore the magnitude of the vectors. As a concrete example, in text classification you may want to ignore whether one document is much longer than another, and just concentrate on the textual content. So in our example above, suppose we have a third document, C, which has seventy occurrences of the word *performance*, thirty occurrences of *transition*, and twenty occurrences of *monetary*. The vector representing C would be C = <70, 30, 20>. If you work through the math you'll find that the cosine distance between A and C is zero—because C is simply A multiplied by 10.

As a final example illustrating the variety of distance metrics, let's again consider text but in a very different way. Sometimes you may want to measure the distance between two strings of characters. For example, often a business application needs to be able to judge when two data records correspond to the same person. Of course, there may be misspellings. We would want to be able to say how similar two text fields are. Let's say we have two strings:

1. `1113 Bleaker St.`
2. `113 Bleecker St.`

We want to determine how similar these are. For this purpose, another type of distance function is useful, called *edit distance* or the *Levenshtein metric*. This metric counts the minimum number of edit operations required to convert one string into the other, where an edit operation consists of either inserting, deleting, or replacing a character (one could choose other edit operators). In the case of our two strings, the first could be transformed into the second with this sequence of operations:

1. Delete a `1`,
2. Insert a `c`, and
3. Replace an `a` with an `e`.

So these two strings have an edit distance of three. We might compute a similar edit distance calculation for other fields, such as name (thereby dealing with missing middle initials, for example), and then calculate a higher-level similarity that combines the various edit-distance similarities.

Edit distance is also used commonly in biology where it is applied to measure the genetic distance between strings of alleles. In general, edit distance is a common choice when data items consist of strings or sequences where order is very important.

* Combining Functions: Calculating Scores from Neighbors

For completeness, let us also briefly discuss "combining functions"—the formulas used for calculating the prediction of an instance from a set of the instance's nearest neighbors.

We began with majority voting, a simple strategy. This decision rule can be seen in Equation 6-6:

Equation 6-6. Majority vote classification

$$c(\mathbf{x}) = \underset{c \in \text{classes}}{\arg\max}\ \text{score}(c, \text{neighbors}_k(\mathbf{x}))$$

Here *neighbors*$_k$($\mathbf{x}$) returns the k nearest neighbors of instance $\mathbf{x}$, *arg max* returns the argument (c in this case) that maximizes the quantity that follows it, and the score function is defined, shown in Equation 6-7.

Equation 6-7. Majority scoring function

$$\text{score}(c, N) = \sum_{\mathbf{y} \in N} [\,class(\mathbf{y}) = c\,]$$

Here the expression *[class($\mathbf{y}$)=c]* has the value one if $class(\mathbf{y}) = c$ and zero otherwise.

Similarity-moderated voting, discussed in "How Many Neighbors and How Much Influence?" on page 149, can be accomplished by modifying Equation 6-6 to incorporate a weight, as shown in Equation 6-8.

Equation 6-8. Similarity-moderated classification

$$\text{score}(c, N) = \sum_{\mathbf{y} \in N} w(\mathbf{x}, \mathbf{y}) \times [\text{class}(\mathbf{y}) = c]$$

where *w* is a weighting function based on the similarity between examples **x** and **y**. The inverse of the square of the distance is commonly used:

$$w(\mathbf{x}, \mathbf{y}) = \frac{1}{dist(\mathbf{x}, \mathbf{y})^2}$$

where *dist* is whatever distance function is being used in the domain.

It is straightforward to alter Equation 6-6 and Equation 6-8 to produce a score that can be used as a probability estimate. Equation 6-8 already produces a score so we just have to scale it by the total scores contributed by all neighbors so that it is between zero and one, as shown in Equation 6-9.

Equation 6-9. Similarity-moderated scoring

$$p(c \mid \mathbf{x}) = \frac{\sum_{\mathbf{y} \in \text{neighbors}(\mathbf{x})} w(\mathbf{x}, \mathbf{y}) \times [\text{class}(\mathbf{y}) = c]}{\sum_{\mathbf{y} \in \text{neighbors}(\mathbf{x})} w(\mathbf{x}, \mathbf{y})}$$

Finally, with one more step we can generalize this equation to do regression. Recall that in a regression problem, instead of trying to estimate the class of a new instance *x* we are trying to estimate some value *f(x)* given the *f* values of the neighbors of **x**. We can simply replace the bracketed class-specific part of Equation 6-9 with numeric values. This will estimate the regression value as the weighted average of the neighbors' target values (although depending on the application, alternative combining functions might be sensible, such as the median).

Equation 6-10. Similarity-moderated regression

$$f(\mathbf{x}) = \frac{\sum_{\mathbf{y} \in \text{neighbors}(\mathbf{x})} w(\mathbf{x}, \mathbf{y}) \times t(\mathbf{y})}{\sum_{\mathbf{y} \in \text{neighbors}(\mathbf{x})} w(\mathbf{x}, \mathbf{y})}$$