Clustering without Probability

36-462/662 Data Mining, Spring 2020

5 March 2020 (Lecture 15)

READING: Hand *et al.* (2001), sections 9.3–9.5.

Contents

1 Categorization and Classifiers

Dividing data into discrete categories is one of the most common kinds of data-mining task. Often the categories are things which are given to us in advance, by some kind of background knowledge (cells: cancerous or not?), or the kind of decision we are going to make (credit applicant: loan them money or not?), or simply by some taxonomy which our institution has decided to use (text: politics or religion? automobile or motorcycle?; pictures: flower, tiger or ocean?). This case, of assigning new data to pre-existing categories, is called **classification**, and the categories are called **classes**.

When we have some examples, or **training data**, which have been **labeled** by someone who knew what they were doing, we have a **supervised** learning problem. The point of classification methods is to accurately assign new, unlabeled examples, from the **test data**, to these classes. This is "supervised" learning because we can check the performance on the labeled training data. The point of calculating information was to select features which made classification easier.

We have already seen a bunch of algorithms for classification. I want to recall two in particular:

1. In **nearest neighbor** classification, we assign each new data point to the same class as the closest labeled vector, or **exemplar** (or **example**, to be slightly less fancy). This uses lots of memory (because we need to keep track of many vectors) and time (because we need to calculate lots of distances), but assumes next to nothing about the geometry of the classes.

2. In **prototype classification**, we represent each class by a single vector, its *prototype*, and assign new data to the class whose prototype is closest. This uses little memory or computation time, but implicitly assumes that each class forms a compact (in fact, convex) region in the feature space.

The appendix to these notes says more about the geometry of these methods, and how it relates to their performance as classifiers.

A more general theme, however, is that statistical and data-mining problems usually have a trade-off between methods which make strong assumptions, and deliver good results when the assumptions hold but break otherwise, and methods which make weak assumptions and work more broadly. One manifestation of this — very much displayed by prototype vs. nearest-neighbor classifiers — is the trade-off between precision and accuracy. (We will say much more about this when we look systematically at how to evaluate and compare predictive models.)

2 Why Cluster?

Classification depends on having both known categories and labeled examples of the categories. If there are known categories but no labeled examples, we may be able to do some kind of **query**, **feedback**, **reinforcement** or learning, if we can check guesses about category membership (or at least be told "warmer" or "colder"). But we might *not* have known classes to start with. In these **unsupervised** situation, one thing we can try to do is to *discover* categories which are in implicit in the data themselves. These categories are called **clusters**, rather than "classes", and finding them is the subject of **clustering** or **cluster analysis**. (See Table 1.)

Even if our data comes to us with class labels attached, it's often wise to be skeptical of their use. Official classification schemes are often the end result of a mix of practical experience; intuition; old theories; prejudice; ideas copied

Known classes?	Class labels	Type of learning problem
Yes	Given for training data	Classification; supervised learning
Yes	Given for some but not all training data	Semi-supervised learning
Yes	Hints/feedback	Feedback or reinforcement learning
No	None	Clustering; unsupervised learning

Table 1: Partial taxonomy of learning problems

from somewhere else; old notions enshrined in forms, databases and software; compromises among groups which differ in interests, ideas and clout; preferences for simple rules; and people making stuff up because they need *something* by deadline. For instance: the threshold for being "overweight", according to official medical statistics, is a body-mass index[1] of 25, and "obesity" means a BMI of 30 or more — not because there's a big qualitative difference between a BMI of 29.9 and one of 30.1, but just because they needed *some* break. Moreover, once a scheme gets established, organizational inertia can keep it in place long after whatever relevance it once had has eroded away. The Census Bureau set up a classification scheme for different jobs and industries in the 1930s, so that for several decades there was one class of jobs in "electronics", lumping together people making radios or thermostats with the whole computer industry[2]. The point being, even when you have what you are *told* is a supervised learning problem with labeled data, it can be worth treating it as unsupervised learning problem. If the clusters you find match the official classes, that's a sign that the official scheme is actually reasonable and relevant; if they disagree, you have to decide whether to trust the official story or your cluster-finding method.

2.1 Good clusters

A good way to start thinking about *how* to cluster our data is to ask ourselves *what* properties we want in clusters. First of all, clusters, like classes, should **partition** the data: every possible object should belong to one, and only one, cluster. Beyond that, it would be good if knowing which cluster an object belonged to told us, by itself, a lot about that object's properties. In other words, we would like the expected information in the cluster about the features to be large. If the features are bundled up in a vector X, and the cluster is C,

[1] $BMI = (mass)/(height)^2$, measuring in kilograms and meters. Physics tells us that at constant density, mass should be proportional to *volume*, rather than *area*, so a more obvious formula would be $(mass)/(height)^3$, which is sometimes called the "ponderal index". The modern use of BMI, $\propto (height)^{(-2)}$, goes back to Keys *et al.* (1972), where the justification was that it correlated a bit better with the actual percentage of fat in the body than either the ponderal index, the simple ratio $(mass)/(height)$, or a variety of other simple formulas that didn't require elaborate measurements of physiological assays. For criticisms of using BMI, especially in "wellness" programs, see O'Neil (2016, pp. 176ff).

[2] For an even more consequential example of how a Census categorical variable was actually made (and re-made, and re-re-made), see Anderson and Fienberg (2000) on how Americans have been classified by race and ethnicity.

we would ideally maximize

$$I[X; C]$$

Actually doing this maximization turns out to be generally hard. However, we can say some things about what the maximally-informative clusters would look like, and use these properties to guide our search.

A high information value for the clusters means that knowing the cluster reduces our uncertainty about the features. All else being equal, this means that the objects in a cluster should be *similar to each other*, or form a **compact** or **homogeneous** set of points in feature-vector space. Again, all else being equal, different clusters should have *different* distributions of features, so clusters should be **separated** or **distinct**. If one of the clusters is much more probable than the others, learning which cluster an object belongs to doesn't reduce uncertainty about its features much, so ideally the clusters should be equally probable, or **balanced**. Finally, we could get a partition which was compact, separated and balanced by saying each object was a cluster of one, but that would be silly, because we want the partition to be **parsimonious**[3], with many fewer clusters than objects.

There are many algorithms which try to find clusters which are compact, separated, balanced and parsimonious. Parsimony and balance are pretty easy to quantify; measuring compactness and separation depends on having a good measure of distance for our data to start with. (Fortunately, similarity search

[3] "Parsimony" comes from the Latin word *parsimonia*, meaning "unwillingness to use resources", coming from a verb *parcere*, "to be sparing" or "to be stingy". The **principle of parsimony** tells us to prefer theories, models or explanations which are simpler to those which are more elaborate, especially to prefer theories which posit fewer unobserved entities — like distinct clusters. (The metaphor is that elaboration is costly, and we're cheap.) The idea, and the name, are often attributed to the medieval scholar William of Occam (or "Ockham"; born sometime around 1287, died 1347, during the Black Plague), so it's sometimes also called "Occam's Razor". (*That* metaphor is that we use the principle to "shave off" superfluous elaborations.) Nobody has been able to find anything called a "razor" in Occam's works, or for that matter "parsimony", though he was certainly fond of attacking opponents for making up complicated explanations without (he thought) good reason; "it is vain to do with more what could be done with less" (Ockham, 1964).

Why *you* should respect the principle of parsimony is a different question. One answer might be that you share Brother Williams's theological views about how God chose to make and govern the world, but those were controversial even among 14th century Catholics (Keele, 2010), and you're probably not a *14th century* Catholic. The principle is sometimes formulated as "the simplest explanation is most likely to be correct", but most ways of making that idea precise are actually wrong (see, e.g., `http://bactra.org/notebooks/occam-bounds-for-long-programs.html`), or question-begging. A very promising idea, advanced by Prof. Kevin Kelly in our philosophy department, is that parsimony helps us learn, even when the truth is complex (Kelly, 2007b,a). Kelly and collaborators have been able to *prove* that the rule "accept the simplest explanation consistent with all the known facts" gives a faster and more direct route to the truth than alternative rules which allow for more complexity than is strictly called for. So even when the truth is complicated (and it often is), the principle of parsimony keeps us from going down dead ends.

(Whether Kelly's idea is what Brother William meant when he said "it is vain to do with more what could be done with less" is a tricky question — my guess is: no, but he might have viewed it as a friendly amendment. But it's a question which is both hard to answer, and quite irrelevant to the *current* merits of the principle of parsimony, so the Razor itself tells us not to bother with the question.)

has taught us about distance!) We'll look first at one of the classical clustering algorithms, and try to see how it achieves all these goals.[4]

3 The k-means algorithm

Recall that in the prototype method, we took the prototype for each class to be its average or mean, and assigned new points to the class with the closest prototype. The **k-means algorithm** is an unsupervised relative of the prototype method for clustering, rather than classification.

Given the number of clusters k and data vectors $\vec{x}_1, \vec{x}_2, \ldots \vec{x}_n$,

1. Randomly assign vectors to clusters
2. Until nothing changes
   (a) Find the mean of each cluster, given the current assignments
   (b) Assign each point to the cluster with the nearest mean

There are many small variants of this. For instance, the R function `kmeans()`, rather than randomly assigning all the vectors to clusters at the start, randomly chooses k vectors as the initial cluster centers, leaves the rest unassigned, and then enters the loop.

To understand how this works, it helps to recall (or learn) an important fact about means. If I take n numbers, $x_1, x_2, \ldots x_n$, their mean is of course defined as

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

But it is also true that

$$\overline{x} = \operatorname*{argmin}_{m} \sum_{i=1}^{n} (x_i - m)^2$$

In words, if I want to approximate the whole collection of numbers by a single number m, I find that the mean minimizes the sum of the squared errors of approximation.[5] This property extends to vectors:

$$\frac{1}{n}\sum_{i=1}^{n} \vec{x}_i = \operatorname*{argmin}_{\vec{m}} \sum_{i=1}^{n} \|\vec{x}_i - \vec{m}\|^2$$

[4] You should however also treat these goals with some suspicion. Take parsimony and balance, for instance. No doubt they're great if the world really does divide into a few evenly-sized clumps, but how often does that happen? For example: One data-set of bird sightings in North America, for instance, contains 591 distinct species; the single most common species accounts for about 7% of all sightings, the least common for 0.00005% (Clauset *et al.*, 2009). Imagine trying to discover the species by clustering the *appearances* of the birds seen. Since the reality is that there are many different kinds of birds, some of which are vastly more common than others, it's hard to see how parsimony and balance will help us.

[5] Of course the mean also minimizes the mean squared error, but that sounds circular.

Now, with k-means, we have k clusters; we'll say these are sets $C_1, C_2, \ldots C_k$, and each vector $\vec{x}_i$ is in one and only one C_j. For each cluster we have a center, $\vec{m}_j$, and a sum of squares,

$$Q_j \equiv \sum_{i:\vec{x}_i \in C_j} \|\vec{x}_i - \vec{m}_j\|^2$$

We can also define $V_j = Q_j/n_j$, with n_j being the number of points in cluster j. This is the **within-cluster variance**.

We have an over-all sum of squares for the whole clustering

$$Q \equiv \sum_{j=1}^{k} Q_j = \sum_{j=1}^{k} n_j V_j$$

If we write a_i for the cluster to which vector i is assigned, and we substitute in the definition of Q_j into that of Q, we see that

$$Q = \sum_{i=1}^{n} \|\vec{x}_i - \vec{m}_{a_i}\|^2$$

the sum of squared distances from points to their cluster centers.

The k-means algorithm tries to reduce Q. In step 2a, we adjust $\vec{m}_j$ to minimize Q_j, given the current cluster assignments. In step 2b, we adjust a_i to minimize Q, given the current means. So at every stage Q either decreases or stays the same — it never grows. We say that Q is the **objective function** for k-means, what it "wants" to minimize. We just saw that Q never increases. Since it's a sum of squares, we know that $Q \geq 0$. This means that Q *must* approach a limit, that it can't keep changing forever. The limit may not be zero and generally won't be, but it must stabilize.

3.1 Geometry of k-Means

The clusters formed by k-means are convex bodies, for the same reason that the prototype method produces convex classification regions. (See the appendix.) This implies that if the true clusters aren't convex regions of feature space, k-means cannot get them right.

We can say a bit more about the geometry of the clusters. Remember that $Q = \sum_j n_j V_j$, with n_j being the number of points in the j^{th} cluster, and V_j the within-cluster variance. If each cluster is compact, it will have a small within-cluster variance, so V_j and Q will be small, which k-means likes. Also, k-means tends to prefer equally-sized clusters, since all else being equal have $n_1 \approx n_2$ makes $n_1 V_1 + n_2 V_2$ smaller. In other words, it favors balance, as discussed above. Finally, to minimize V_j around a given center, the points should be arranged in a circle (sphere, hyper-sphere) around that center, so the ideal cluster for k-means is a rounded blob.

3.2 k-Means as Search Algorithm

K-means is a **local search** algorithm: it makes small changes to the solution that improve the objective. This sort of search strategy can get stuck in **local minima**, where the no improvement is possible by making small changes, but the objective function is still not optimized.

It's often helpful to think of this in terms of a **search landscape**, where the height of the landscape corresponds to how good a solution the algorithm has found. (So *minimizing* the objective function is the same as *maximizing* the height on the landscape.) Local search is also called **hill climbing**, because it's like a short-sighted climber who tries to get to the top by always going uphill. If the landscape rises smoothly to a central peak, this will get to that peak. But if there are local peaks, it can get stuck at one, and which one it reaches depends on where the climb starts.

For k-means, the different starting positions correspond to different initial guesses about the cluster centers. Changing those initial guesses will change the output of the algorithm. These are typically randomized, either as k random data points, or by randomly assigning points to clusters and then computing the means. Different runs of k-means will thus generally give different clusters, but you can actually make use of this: if some points end up clustered together in many different runs, that's a sign that they really do belong together.

3.3 k-Means for Nearest Neighbors

One cute use of k-means is to speed up nearest-neighbor prediction[6]. Implementing nearest neighbors in the most straightforward way means finding the distance between the point where we want the prediction, say x_0, and all of the n training points in our data set, say $x_1, \ldots x_n$. Since there are n distances, this needs $O(n)$ calculations. We then have to find the smallest distance, but this is at worst $O(n)$ again, and possibly better if we're clever[7]. We then need to average the observed responses of the j nearest neighbors, which takes $O(j)$ calculations. So making one prediction with straightforward j-nearest-neighbors takes $O(n + j)$ time. Since $n \gg j$, the vast majority of that time and effort is spent calculating distances to training points we don't end up using!

Clearly, something faster would be nice, and a lot of thought has gone into fast nearest neighbor algorithms, including ones for finding *approximate* nearest neighbors quickly. Some of these algorithms involve clever data structures, like "k-d trees" (Bentley, 1975) (which the `FNN` library implements). A more recent approach is to use clustering, and specifically to use k-means (Paulevé *et al.*, 2010). Here's how it goes in its simplest form:

[6] It's a little unfortunate that the names are "k-means" and "k-nearest-neighbors", because the ks mean different things. So in this subsection, we'll say it's "j-nearest-neighbors"

[7] For instance, we could keep a list of distances to the n training points sorted in increasing order as we calculate each distance, which can be done quickly. Then we take the first j points in the list.

1. Take the training data and create k clusters by running k-means, say $k = O(\sqrt{n})$.

2. When we need to make a prediction for a new point x_0, find the cluster with the closest center, and look for its neighbors *only* among training points in that cluster.

To see how this can help, notice that the average number of training points per cluster is $n/k = n/O(\sqrt{n}) = O(\sqrt{n})$. So finding the closest cluster to x_0, in step 2, takes $O(\sqrt{n})$ time (because that's how many clusters there are), and finding the nearest neighbors to x_0 within that cluster takes $O(\sqrt{n})$ time (because that's how many points are in the cluster), for an over-all time[8] of $O(\sqrt{n}) + O(\sqrt{n}) = O(\sqrt{n})$, which is a *lot* better than $O(n)$ for big n (say 10^6 or 10^9). Moreover, we're not restricting our search for neighbors to just some random set of $O(\sqrt{n})$ data points. We know that k-means tends to produce compact clusters of points which are all close to each other, and they're generally close to the new point x_0 because that's why we assigned the point to *this* cluster. So the points in that cluster are good bets for being the nearest neighbors, or close to the nearest neighbors[9]. This doesn't always work — you can design counter-examples where the nearest neighbors to x_0 will all be in different clusters — but it often works really well.

3.4 Some Fast Extensions

The classic k-means algorithm assigns the initial clusters randomly. If you really want to minimize the sum of within-cluster distances, you can often do much better by choosing the initial clusters intelligently, and then running k-means as usual. The classic paper on this is Arthur and Vassilvitskii (2007); I'm not sure if there's an R implementation.

As it says in the name, k-means defines the center of each cluster as its arithmetic mean. You could swap in any measure of central tendency, so we have k-medians (for instance), or k-trimmed-means, etc. Many of these are more robust to outliers than the arithmetic mean, so they can give nicer-looking results in the face of heavy-tailed or contaminated data. (How would you prove that k-medians converges?)

Relatedly, in any space where we have a notion $d(x, y)$ of the distance between points x and y, we can follow the great French mathematician M. Frechet

[8] Of course, it takes some time to run k-means initially, but if we're making a lot of predictions, that initial set-up time is shared among, or "amortized over", all the predictions, rather than growing with the number of predictions.

[9] A modification which improves the odds of finding the real nearest neighbors this way is to run k-means multiple times, from multiple random seeds, say l times. This gives l different partitions. We find which cluster x_0 will be assigned to in each partition, and then combine those data points for our detailed search for the j nearest neighbors. Finding the cluster assignment of x_0 over the l partitions takes $O(l\sqrt{n})$ time, and finding the j nearest neighbors takes $O(l\sqrt{n})$ time (or less; hopefully lots of points get clustered together with x_0 across multiple partitions), so the over-all time is $O(l\sqrt{n})$.

and define the **Frechet mean**[10] of the set $x_1, x_2, \ldots x_n$ as

$$\operatorname*{argmin}_y \sum_{i=1}^{n} d^2(x_i, y)$$

For ordinary vectors, these are just common or garden arithmetic means. But this gives us a notion of "mean" which will work even in spaces where it *doesn't* make sense to take averages, like spaces of categorical variables or sequences of categorical variables, so long as we can define distances between points. So k-means generalizes naturally to k-Frechet-means.

There are also many algorithmic variants. For example, we might re-assign *one* point to a new cluster, and then re-calculate the cluster centers (how many would we have to adjust?), rather than switching between updating all the cluster assignments and all the cluster centers. Kogan (2006) has a good discussion, with references.

4 Distances Between Partitions

Different clustering algorithms will give us different results on the same data. The *same* clustering algorithm may give us different results on the same data, if, like k-means, it involves some arbitrary initial condition. We would like to say how far apart two clusterings of the same data are. In doing this, we want to accommodate different numbers of clusters, and we want to accommodate the fact that cluster labels are completely arbitrary, so we want to say two clusterings are the same if they *only* differ by the labels.

Recall that a **partition** of a set divides it into subsets where are **mutually exclusive** (no point in the set is in more than one subset) and **jointly exhaustive** (every point is in some subset). The subsets are called the **cells** of the partition. When we have class labels, the classes partition the data. What we get from a clustering procedure is another partition.

Whenever we have two partitions of the same data, we can build a **confusion matrix**, just as we did for classifiers; call it A. The entry A_{ij} is the number of items which are in cell i of the first partition and cell j of the second partition. When we did this for classifiers, the first partition was that of the true classes, and the second partition was that of our guesses; we wanted the diagonal entries of A to be big and the rest small, because that meant we were guessing correctly. Two clusterings can be close, however, even if the diagonal isn't, because the numbering of the clusters is essentially arbitrary.

One distance measure which does what we want — which is invariant under permutations of the cluster labels — is what's called the **variation of information** metric (Meila, 2003).[11] Pick a point from the set completely at random, and let C be the cell it falls into according to the first partition, and D its cell

[10] Frechet didn't call them Frechet means.

[11] It has many names, actually.

in the second partition. Then the distance is

$$H[C|D] + H[D|C] \tag{1}$$

This will be zero if and only if there is a 1-1 correspondence between the cells of the two partitions. Otherwise, it will be positive, and the larger it is, the less information we get about one partition from the other. (Its maximum possible value is $H[C] + H[D]$, and some people scale the distance to $[0, 1]$ by dividing by this.

5 Further Reading

As mentioned in class, clustering is harder than classification, because there are *necessarily* competing objectives, requiring decisions and trade-offs. I discussed the idea that a clustering should be a partition, so that each point belongs to one and only one cluster. Next time, we'll look at hierarchical clustering, where each point belongs to many clusters, because clusters themselves are parts of bigger clusters. We'll also go on to look at probabilistic clusters, where we can never say for sure which cluster a point came from, and mixed-membership models, where each point has a *share* in multiple groups or clusters.

I also mentioned that there are proofs that we can't reconcile all the objectives we might want from clustering. The most important such result goes back to Kleinberg (2002). He considered clustering algorithms which worked with the distances between points to produce partitions (as does k-means), and proved that *no* procedure could combine three natural, harmless-seeming properties:

1. The algorithm can generate all possible partitions of n points;
2. Multiplying all the distances by the same factor doesn't change the partition we get;
3. Bringing all the points in the same cluster closer together, and all the points in different clusters further apart, doesn't change the partition.

Giving up the first property means our procedure is ruling out some clusterings without even looking at the data; we just don't believe it. (Why not, if we're doing clustering to begin with?) Giving up the second property means we'd get different clusters if we measure distances in inches than in centimeters. (Who died and left our units of measurement in charge?) Giving up the third property also sounds weird, but it *might* be a little more acceptable[12]. People have nibbled around the edges of this impossibility result[13], but the point stands.

[12] E.g., what if we bring all the points within some cluster closer together, but in the process split them into two lumps of *really* tightly bunched-up points?

[13] For instance, Ackerman and Ben-David (2009) show how to define measures of how good a cluster is that meet all 3 of Kleinberg's requirements.

Exercises

To think through, not to turn in.

1. Prove Eq. 3. (There are many ways to do this; the brute-force approach is to differentiate.)
2. Write your own k-means clustering algorithm. You may want to use the prototype classifier from the homework.
3. Give an example of two partitions (of the same set of points) where $H[C|D] = 0$ but $H[D|C] > 0$. Can you do this where both the C and the D partition have the same number of cells?

A Geometric Aspects of the Basic Classifiers

This appendix goes over some of the geometry of the two basic classifier methods, and how this geometry connects to their statistical properties.

A.1 Prototype Method

Start with prototype classification. For each class i, $i = 1, \ldots k$, we have a prototype vector ρ_i. We assign a new vector x to the class with the closest prototype:

$$R(x) = \underset{i}{\operatorname{argmin}} \|x - \rho_i\| \tag{2}$$

In other words, $R(x) = i$ if, and only if, $\|x - \rho_i\| \leq \|x - \rho_j\|$, $j \neq i$.

To see what this means, suppose that there are only two classes (i.e., $k = 2$). Then

$$R(x) = \begin{cases} 1 & \|x - \rho_1\| \leq \|x - \rho_2\| \\ 2 & \|x - \rho_1\| > \|x - \rho_2\| \end{cases}$$

(This arbitrarily breaks the tie exactly on the boundary in favor of class 1. How we break the tie is immaterial.) What *that* means, as I drew on the board, is that the prototype method, in effect, draws the line segment from ρ_1 to ρ_2, and then draws the perpendicular bisector of that line. (In d dimensions, the perpendicular bisector is a $(d-1)$-dimensional hyperplane.) Everything on one side of the bisector goes into class 1, and everything on the other side goes into class 2. This divides the space into two **half-spaces**. Each half-space is a **convex set**, meaning that if x and y are both in the set, every point on the line-segment connecting them is also in the set.

Now suppose there are more than two classes. The vector x gets assigned to class i if, and only if, it would be assigned to class i in every two-way match-up against the other classes. This means that the set of points which are assigned to the class i is the intersection of all of those half-spaces. Each half-space is a convex set, and the intersection of any number of convex sets is another convex set[14] Thus, the prototype method represents the classes by convex sets (in fact by convex polygons, or their generalizations to higher dimensions). This will work well if the classes can, in fact, be well-approximated by convex sets. Examples like the bulls-eye pattern show situations where this implicit assumption of the prototype method fails spectacularly.

The advantages of the prototype method are, once again, the limited amount of information to be stored (k prototype vectors), the fast computation to be performed (k distances calculations, and finding the smallest number from a list of k numbers), and the speed with which it converges to a final answer.

[14] A quibbler might ask, What if the two convex sets don't overlap? Then their intersection is the empty set, which is technically convex. However, we know that ρ_i is closer to *itself* than is any other point, so all the half-spaces for class i have at least one point in common, and their intersection is therefore not empty.

A.1.1 Rate of Convergence

Suppose that data vectors $X_1, X_2, \ldots$ are produced by independent and identically-distributed samples, and that the class C_i of a data-point depends deterministically on the vector. That is, $C_i = c(X_i)$ for some function $c(\cdot)$. (We will see later how to handle the case where the feature vector doesn't have enough information to perfectly classify cases.) For each class, then, there is a true mean vector:

$$\rho_i^* = \mathbb{E}\left[X | c(X) = i\right]$$

We don't know the true means, but we can approximate them from our sample, and uses those approximations as our ρ_i:

$$\rho_i = \frac{\sum_{k=1}^{n} X_k \mathbf{1}_i C_k}{\sum_{k=1}^{n} \mathbf{1}_i C_k}$$

(This is just the mean of all the sample vectors whose class is i.)

With IID samples, the law of large numbers tells us that $\rho_i \to \rho_i^*$. More than that, the central limit theorem tells us that $|\rho_i - \rho_i^*\| = O(n^{-1/2})$. Since the locations of the estimated centers determine the boundaries and so the classifications, and the estimated centers converge rapidly, the classification function $R(x)$ will converge rapidly to a limiting function.[15]

A.2 Nearest-Neighbor Method

In the nearest-neighbor method, we have a set of data vectors $x_1, x_2, \ldots x_n$, each of which already has a class label, $c_1, c_2, \ldots c_n$. We then assign a new vector x to the class of its nearest neighbor:

$$\begin{aligned} N(x) &= \operatorname*{argmin}_i \|x - x_i\| \\ C(x) &= c_{N(x)} \end{aligned}$$

Each of our example vectors x_i thus has its own region of feature-space where it decides the class of new vectors. The analysis above for prototypes carries over: it's the intersection of all the half-spaces which contain x_i and divide it from the other points. So each x_i gets its own convex set where it decides the class of new vectors. Since (in general) multiple examples have the same class label, each class corresponds to the *union* of several convex sets in feature space, and the union of convex sets is not (in general) convex. This gives nearest-neighbor classification more flexibility or **expressive power** than the prototype method.

The price for the extra expressive power is paid in memory (in general, the algorithm needs to keep track of all the examples), in computational time (to calculate distances and find the shortest distance) and in the rate of convergence.

[15] Unless some of the true class means are equal, i.e. $\rho_i^* = \rho_j^*$ for some $j \neq i$. Then i and j will end up trying to divide the same cell of the partition, in an essentially random way that just reflects the difference in their sampling fluctuations.

With (as before) IID samples, the performance of the nearest neighbor rule is ultimately very good: if (as before) the class is a deterministic function of the features, $C_i = c(X_i)$, then in the long run the error rate of the nearest neighbor classifier goes to zero. More generally, if the class is only probabilistically related to the features, then the error rate converges to no more than *twice* the best attainable rate, as we've seen in an earlier lecture (Cover and Hart, 1967). As for the *rate* of convergence, the probability of error *can* shrink like $1/n^2$, but it can also shrink arbitrarily slowly (Cover, 1968).

References

Ackerman, Margareta and Shai Ben-David (2009). "Measures of Clustering Quality: A Working Set of Axioms for Clustering." In *Advances in Neural Information Processing Systems 21 [NIPS 2008]* (Daphne Koller an d D. Schuurmans and Yoshua Bengio and Léon Bottou, eds.), pp. 121–128. Curran Associates. URL `https://papers.nips.cc/paper/3491-measures-of-clustering-quality-a-working-set-of-axioms-for-clustering`.

Anderson, Margo and Stephen E. Fienberg (2000). "Race and Ethnicity and the Controversy over the US Census." *Current Sociology*, **48**: 87–110. doi:10.1177/0011392100048003007.

Arthur, David and Sergei Vassilvitskii (2007). "`k-means++`: The Advantages of Careful Seeding." In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms [SODA07]* (Harold Gabow, ed.), pp. 1027–1035. Philadelphia: Society for Industrial and Applied Mathematics. URL `http://www.stanford.edu/~darthur/kMeansPlusPlus.pdf`.

Bentley, Jon Louis (1975). "Multidimensional binary search trees used for associative searching." *Communications of the ACM*, **18**: 508–517. doi:10.1145/361002.361007.

Clauset, Aaron, Cosma Rohilla Shalizi and M. E. J. Newman (2009). "Power-law Distributions in Empirical Data." *SIAM Review*, **51**: 661–703. URL `http://arxiv.org/abs/0706.1062`.

Cover, Thomas M. (1968). "Rates of Convergence for Nearest Neighbor Procedures." In *Proceedings of the Hawaii International Conference on Systems Sciences* (B. K. Kinariwala and F. F. Kuo, eds.), pp. 413–415. Honolulu: University of Hawaii Press. URL `http://www-isl.stanford.edu/~cover/papers/paper009.pdf`.

Cover, Thomas M. and P. E. Hart (1967). "Nearest Neighbor Pattern Classification." *IEEE Transactions on Information Theory*, **13**: 21–27. URL `http://www-isl.stanford.edu/~cover/papers/transIT/0021cove.pdf`.

Hand, David, Heikki Mannila and Padhraic Smyth (2001). *Principles of Data Mining*. Cambridge, Massachusetts: MIT Press.

Keele, Rondo (2010). *Ockham Explained: From Razor to Rebellion*. Chicago, Illinois: Open Court.

Kelly, Kevin T. (2007a). "A New Solution to the Puzzle of Simplicity." *Philosophy of Science*, **74**: 561–573. doi:10.1086/525604.

— (2007b). "Ockham's razor, empirical complexity, and truth-finding efficiency." *Theoretical Computer Science*, **383**: 270–289. doi:10.1016/j.tcs.2007.04.009.

Keys, Ancel, Flaminio Fidanza, Martti J.Karvonen, Noboru Kimura and Henry L.Taylor (1972). "Indices of relative weight and obesity." *Journal of Chronic Diseases*, **25**: 329–343. doi:10.1016/0021-9681(72)90027-6.

Kleinberg, Jon (2002). "An Impossibility Theorem for Clustering." In *Advances in Neural Information Processing Systems 15 [NIPS 2002]* (Suzanna Becker and Sebastian Thrun and Karl Obermayer, eds.), pp. 463–470. Cambridge, Massachusetts: MIT Press. URL `https://papers.nips.cc/paper/2340-an-impossibility-theorem-for-clustering`.

Kogan, Jacob (2006). *Introduction to Clustering Large and High-Dimensional Data*. Cambridge, England: Cambridge University Press.

Meila, Marina (2003). "Comparing Clusterings by the Variation of Information." In *Computational Learning Theory and Kernel Machines: 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop [COLT/Kernel 2003]* (Bernhard Schölkopf and Manfred K. Warmuth, eds.), pp. 173–187. Berlin: Springer. URL `http://www.stat.washington.edu/mmp/Papers/compare-colt.pdf`. doi:10.1007/978-3-540-45167-9_14.

Ockham, William of (1964). *Philosophical Writings: A Selection, Translated, with an Introduction, by Philotheus Boehner, O.F.M., Late Professor of Philosophy, The Franciscan Institute*. Indianapolis: Bobbs-Merrill. First pub. various European cities, early 1300s.

O'Neil, Cathy (2016). *Weapons of Math Destruction: How Big Data Increases Inequality and Threatens Democracy*. New York: Crown.

Paulevé, Loïc, Hervé Jégou and Laurent Amsaleg (2010). "Locality sensitive hashing: a comparison of hash function types and querying mechanisms." *Pattern Recognition Letters*, **31**: 1348–1358. URL `https://hal.inria.fr/inria-00567191/document`. doi:10.1016/j.patrec.2010.04.004.