Machine Learning Concepts

September 5, 2022

Now that we understand some of the perils of polynomial data fitting in the large sample spaces involved in machine learning, we revisit some of the basic concepts we have introduced quite informally so far. These include different types of machine learning problems, the concept of *learning* a regressor or a classifier from data, and the curse of dimensionality.

1 Supervised and Unsupervised Learning

Machine learning develops algorithms that discover patterns in data. In addition to the problems mentioned in the last set of notes, we consider the following examples of classification, regression, and clustering, with the aim of tightening some of the basic definitions and introducing some notation we will use throughout the course.

Examples:

- **Classification:** The US Postal Service (USPS) uses digit recognition, a machine learning technique, to read handwritten ZIP codes on envelopes. Many thousands or even millions of images of the digit '0' are provided, together with the label '0', and similarly for the other nine digits. These images are provided by the USPS, and someone painstakingly looks at each image and records a label for it in a file. The machine learning algorithm must identify some commonalities among all images of the same digit (these commonalities are the "pattern" in the data) so that if a previously unseen image shows up without a label, it can be automatically labeled without human intervention. The way the machine learning algorithm captures the "pattern" in practice is by computing a function that takes an image as input and produces the appropriate label as output.
- **Regression:** YouTube may be interested in an automatic method to predict the median age of the viewers of each given video clip, so that it can target ads it shows with each video more specifically and cost-effectively. In this scenario, YouTube may collect data through surveys, perhaps by asking viewers to state their ages voluntarily when they access a new clip. Individual responses may not be very reliable, but if there are enough of them per video clip, the empirical median might be an accurate summary. The machine learning algorithm must identify some pattern that connects properties of each video to the median age of its viewers. Once that pattern is found, it can be used to predict median ages of viewers without including the annoying survey.
- **Clustering:** When a color image is to be sent over an expensive channel or displayed on an inexpensive device, it may not be possible to preserve all the original colors (because doing so would be too expensive, in the first case, or because the device is unable to, in the second). A common solution is to fix the number K of allowable colors to some small integer, and then map each color in the original image to the most similar of the K allowable ones. This requires building a *colormap*, that is, a table of K allowable colors. Different images have different color distributions, and therefore may require different colormaps for optimal rendering.

  This *color quantization* problem can be viewed as one of *clustering* if the colormap is computed from a subset of the pixels in the original image (as is typically done to save computation effort) and is then applied to all the pixels: The N colors $\mathbf{x}_1, \ldots, \mathbf{x}_N$ in the sample are viewed as an *unlabeled* training set X (not T, because there are no labels), and the goal of clustering is to group these colors into $K \ll N$ groups such that colors within a group are similar to each other and colors in different groups are different from each other. Once this task is accomplished, one can select a representative color for each group to form the colormap, and map each of the colors in the original

image (typically many more than N) to the nearest of the K representatives (in a metric suitable for colors). Here, the "pattern" in the data is some natural grouping of colors.

These examples stem from image or video analysis problems because my main area of research is computer vision. There are many more application domains for machine learning.

1.1 Classification

Let us formalize what happens in the classification example above: The machine learning algorithm takes a *training set* as input, of the form

$$T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\} \quad \text{with} \quad \mathbf{x}_n \in X \quad \text{and} \quad y_n \in Y \ ,$$

where each *data point* $\mathbf{x}_n$ is an image and the corresponding y_n is that image's *label* (a digit between 0 and 9 in the example). Each of the thousands or millions of ordered pairs $(\mathbf{x}_n, y_n)$ is called a *training sample*. The set X is the *data space*, and in the example it is the set of *all possible* images (of some appropriate format), not just the set of training data points. The *label set* Y is the set of all digits

$$Y = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

in the example. The fact that the elements of Y are numbers (in this case) is irrelevant. Think of labels as abstract identifiers.

The output of the machine learning algorithm is a *function* h that takes an image $\mathbf{x}$ and returns a label y, that is,[1]

$$h \ : \ X \to Y \quad \text{such that} \quad h(\mathbf{x}) = y \ . \tag{1}$$

This function is the *classifier* or *predictor* learned by the machine learning algorithm. Importantly, while the classifier h "works" even if $\mathbf{x}$ is one of the training samples $\mathbf{x}_n$ (and hopefully yields the appropriate label y_n at least most of the time), in real life h will be typically applied to new, previously unseen images $\mathbf{x}$ for which no labels are available, and produce labels y for them that are hopefully correct most of the time: All the manual work that goes into taking and labeling images in order to make the training set T pays off in the form of a classifier h that can do the labeling work for you from now on. The machine learning algorithm somehow found the "pattern" in the labeled digit images (the "data"), and captured that pattern in the classifier h.

For computer scientists, there is little difference between an algorithm and a function: Every algorithm that takes inputs and produces outputs can be represented by a mathematical function, and every mathematical function can be implemented, at least approximately, by some algorithm. Thus, the machine learning algorithm itself can be viewed as a function that takes a set T as input and produces a function h as output. In order to do so, the algorithm needs to know what functions h with the signature given in expression 1 it is allowed to choose from. The set $\mathcal{H}$ of such functions is called the *hypothesis space* for the given learning problem.[2]

Think, if you can, of the class

$$\mathcal{T} \stackrel{\text{def}}{=} 2^{X \times Y}$$

of all possible training sets.

[1] The "right arrow" description of a function h in terms of its domain X and codomain Y, as given in expression 1, is called the *signature* of the function.

[2] For this and other terminology in machine learning, we defer to common use in the literature. All terms have some historical justification, even when they sound a bit strange.

Notation: The Cartesian product $X \times Y$ is the set of all possible data-point/target pairs, that is, the set of all possible data samples (whether they are in a specific training set or not). If A is some set, the expression 2^A denotes[3] the *power set* of A, that is, the class of all subsets of A. Thus, $\mathcal{T}$ is the class of all possible sets of training samples.

This class and $\mathcal{H}$ are very large classes indeed, and we only think about them abstractly: We will never be asked to produce all of their elements. Then, the signature of the machine learning function λ is

$$\lambda \ : \ \mathcal{T} \to \mathcal{H} \quad \text{such that} \quad \lambda(T) = h$$

and we say that λ *learns* or *trains* the classifier h from the training set T. Applying a classifier to labeled data unseen during training is called *testing* the classifier. Applying a classifier to any data is called *inference*.

If a function that outputs a function makes your head spin, think of both λ and h as algorithms: You write an algorithm λ that somehow looks at the training samples in the training set T and produces a piece of code h. That piece of code can be run on new images $\mathbf{x}$ to guess their labels, $y = h(\mathbf{x})$.

Even this "algorithmic" view is a bit abstract. What happens most of the time is that the machine learning algorithm outputs the vector $\boldsymbol{\theta}$ of *parameters* of a parameterized algorithm $h_{\boldsymbol{\theta}}$. Think of $h_{\boldsymbol{\theta}}$ as a piece of code that is known ahead of time (we write that code), but that is missing the values of some parameters $\boldsymbol{\theta}$. The machine learning algorithm λ knows about $h_{\boldsymbol{\theta}}$, and only outputs the vector $\boldsymbol{\theta}$. Hopefully this more concrete view of things eases your anxiety. Mathematically, it is notationally simpler and conceptually more general to think of λ as producing the whole function h.

It is useful to observe that if the label set Y has K labels, a classifier defines a *partition* of X into K subsets $X_1, \ldots, X_K$ by the following rule[4]:

$$\mathbf{x} \in X_y \ \Leftrightarrow \ h(\mathbf{x}) = y \quad \text{for} \quad y \in Y \ .$$

Thus, another way to view the process of learning a classifier is that the training algorithm λ partitions the set X of all possible inputs (not just the set of training data points!) into K subsets, with each subset assigned to one of the labels. The classifier h is a piecewise-constant function, and the "pieces" are the subsets X_y.

Practical Aspects: Making a Training Set. It is useful to examine what it takes to produce the images $\mathbf{x}_n$ for the classification example above. One could place a camera above one of the conveyer belts that move envelopes around in a USPS distribution center, and take a picture of each envelope. That picture, however, contains much more than a digit, and someone must therefore sit at a computer, display each envelope image on the screen in turn, drag a bounding box around each digit with a mouse, and save each cropped digit as a separate file. Another file will record the name of the image and the corresponding label (what digit the image represents). This collection of data is the training set T.

Different images may have different quality depending on factors such as lighting, the ink used to write the digit, the color of the envelope paper, the size of the digit, and so forth. In addition, the digit images may be cropped inconsistently if the operator gets tired, or multiple operators are employed to crop and label images. These inconsistencies often make learning harder, and training sets are sometimes *curated* because of this. In the USPS example, curation involves running manual procedures to make the images more uniform in terms of the factors mentioned above.[5] Curation helps machine learning research by making the problem easier

[3] This is fitting notation, because if A has a finite number k of elements, then 2^A has 2^k elements.

[4] The double arrow reads "if and only if."

[5] Think calibrating, touching up, and re-cropping or resizing images in Photoshop.

for initial study. However, curation cannot be used to train real-life machine learning classifiers, because the whole point of computing h is to eliminate manual intervention during deployment (testing). In these cases, only automatic *preprocessing* of the images is a viable option, because the preprocessing can then be applied during both training and testing. Preprocessing techniques are beyond the scope of these notes.

In the days of Google Images and web crawlers, collecting data for training is often inexpensive if no curation is needed. Labeling, on the other hand, is laborious, time-consuming, and error-prone. A common method for labeling large amounts of data is to publish it to the Amazon Mechanical Turk, an online marketplace for the type of repetitive work involved in labeling. People around the world access the marketplace searching for easy jobs, and you pay them a small amount (often 1-3 cents) per label. Even so, labels are not always of good quality, and various schemes are devised to address this issue. For instance, multiple people could be employed to label each training sample, and a majority vote could then be taken to determine the most likely label. In any case, the cost of labeling is high, and many machine learning algorithms require large amounts of data. This high cost is arguably the main hurdle to a broader use of machine learning.

1.2 Regression

Formally, the problem posed in the YouTube regression example given earlier is nearly identical to the classification problem: Discover a pattern in the training data (video clips and corresponding recorded median viewer ages) to learn a function h that predicts median ages for previously unseen (and unlabeled) video clips. As a result, most of the considerations and terminology from the previous section still hold.

The difference between classification and regression is that the function h outputs a *categorical* variable for classification and a *real-valued* variable for regression. This means that the label set Y is finite and unstructured for classification, while for regression we have

$$Y = \mathbb{R}\,,$$

the set of real numbers, endowed with a metric (it makes sense to talk about the "distance" between two ages). Incidentally, while a natural distance exists for digits, the distance is irrelevant for classification: Knowing that $|2-3|$ is smaller than $|2-5|$ does not help distinguish the two digits from each other. Thus, any distance that may exist in Y is not *used* in classification, even it there happens to be a natural one.

In terms of definitions, the difference between classification and regression is minor. In practice, however, the finite nature of the label set often suggests techniques for classification that could not be applied to regression. Conversely, regression techniques typically rely on the chosen metric for Y, and these techniques cannot be used for classification. The separation between regression and classification is not always strong, as some machine learning algorithms can be applied to either problem with minor changes.

The predictor h output by a machine learning algorithm for regression is called a *regressor*, rather than a classifier, and the word *value* is used for Y instead of label. All other considerations made earlier for classification hold for regression as well. A *target* is either a label or a value.

1.3 Unsupervised Learning

Classification and regression are said to be *supervised* machine learning problems, with targets constituting the supervision. In other words, the machine learning algorithms for these problems

are told explicitly (by a "supervisor") what target is to be associated to each data sample in the training set.

The clustering problem described earlier, on the other hand, is an example of *unsupervised learning*, because the inputs come with no targets. Instead, the number K of desired clusters is (usually but not always) given as an input in addition to the set

$$T = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$$

of unlabeled data points (the colors in the sample). Part of the desired output of the machine learning algorithm λ is again equivalent to a classifier

$$h \ : \ X \to Y$$

where X is the set of all possible colors and

$$Y = \{1, \ldots, K\} \ ,$$

a set of indices into the table of allowed colors. This function takes an original color $\mathbf{x}$ (inside or outside the training set T) and maps it to color number y in the colormap. In this context, the function h is called an *indicator function*, rather than a classifier, and defines a *partition* of X into K subsets $X_1, \ldots, X_K$ by the following rule[6]:

$$\mathbf{x} \in X_y \ \Leftrightarrow \ h(\mathbf{x}) = y \quad \text{for} \quad y \in Y \ ,$$

just as for classification.

However, in addition to h, the algorithm λ must also output the colormap, that is, a mapping

$$c \ : \ Y \to X$$

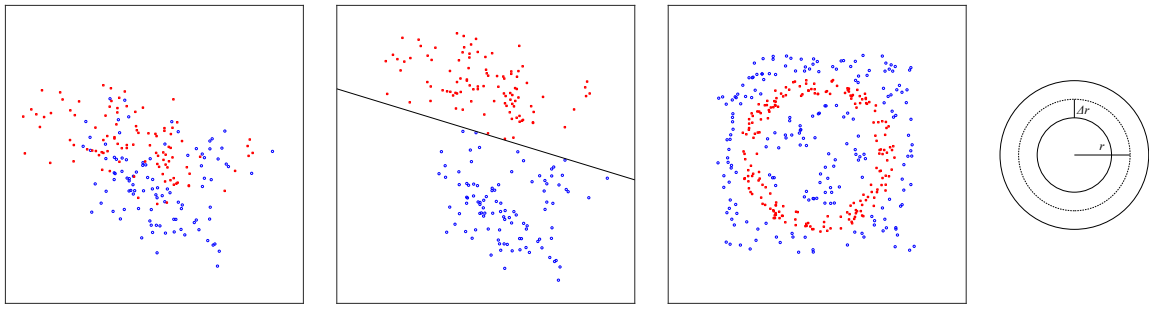
which tells which color corresponds to which index in Y. The colors $c(y)$ in the map are not necessarily colors that are present in the original image.

What makes this machine learning task a *clustering* problem is that h must be constructed so that the partition of X it induces must be "natural:" colors within a subset X_y must be more similar to each other than colors in different subsets.

In turn, this constraint requires the existence of a *distance* (or, conversely, a *similarity*) function, that is, a measure for how different (or similar) two colors are. This constraint also requires defining some clustering *loss*, that is, a function that measures how bad a given clustering h is for some input set T. The learning algorithm then finds h so as to minimize the average loss on T (perhaps approximately).

Obviously, different distance or similarity functions and different loss functions yield different clusterings, and there are no "universal" choices for these aspects. We will *not* cover unsupervised learning in this course, and we therefore refrain from further elaboration or formalization, except to note that color quantization is really a "miniature" example of a machine learning problem: Each image is a new learning problem with its own training set T. In other words, while λ is typically run once in a classification problem, and h is then deployed in the real world, the function being deployed in color quantization is λ itself, and a new indicator function h is learned for each new image. Thus, each new image involves both training and inference.

[6]The double arrow reads "if and only if."



(a) (b) (c) (d)

Figure 1: (a) A graphical representation of a training set with 200 samples belonging to two classes. The data points (features) $\mathbf{x}_n$ are pairs of real numbers. The blue circles have label $y_n = \texttt{'c'}$ and the red squares have label $y_n = \texttt{'s'}$, so each sample $(\mathbf{x}_n, y_n)$ is an element of $X \times Y$ where $X \subseteq \mathbb{R}^2$ and $Y = \{\texttt{'c'}, \texttt{'s'}\}$. (b) A different, linearly separable training set with 200 samples. (c) A training set that is not linearly separable but has a simple boundary between the two classes. (d) Definition of radius r and half-width Δr of a ring centered at the origin of the plane.

1.4 Drawings and the Curse of Dimensionality

We often draw pictures of machine learning problems and datasets on the plane (of a sheet of paper, a computer screen, or a whiteboard). Each data point $\mathbf{x}$ is a point on that plane, and the boundaries of the partition induced by a classifier are sketched with lines, straight or curved depending on the type of classifier. For instance, Figure 1 (a) shows data that belong to two classes, the blue circles and the red squares. Finding a classifier amounts to finding a set of curves that partitions the plane into two regions, not necessarily connected, such that all the blue circles are in one region and all the red squares are in the other.

From this drawing, one comes away with the impression that these curves are quite complicated, and one is then led to seek a large hypothesis space $\mathcal{H}$, in the hope that it will contain a function $\hat{h}$ that achieves the desired separation. This function would take value `'s'` on the region with the red squares and value `'c'` on the rest of the plane, and the separating curves are where $\hat{h}$ switches values.

If, on the other hand, the drawing looks like the one in Figure 1 (b), there is a straight line that (barely) separates the two sets. This is an example of a *linearly separable* training set T. For this problem, the hypothesis space $\mathcal{H}$ could be much simpler. For instance, one could define some set of *scoring functions* $s_\ell(\mathbf{x})$ parameterized by a line ℓ on the plane, and with the property that $s_\ell(\mathbf{x}) > 0$ on one side of ℓ and $s_\ell(\mathbf{x}) < 0$ on the other. The classifier is then a *score-based classifier*,

$$h_\ell(\mathbf{x}) = \begin{cases} \texttt{'s'} & \text{if } s_\ell(\mathbf{x}) > 0 \\ \texttt{'c'} & \text{otherwise} \end{cases}$$

and finding the optimal classifier $\hat{h}$ amounts to finding the appropriate line ℓ (with the appropriate orientation as well, so that the red squares are on the side of the line where the score is positive). We will see examples of scoring functions of this type later in the course.

Even if the data is not linearly separable, it may fail to be separable for rather secondary reasons of representation. For instance, the data in Figure 1 (c) is not linearly separable. However, all the

red squares are inside a ring, and all the blue circles are outside it. Let us assume that the center of the ring is the origin of the plane, $(0, 0)$. The hypothesis space $\mathcal{H}$ could be tailored to this type of data, and include all binary functions that have one value in some ring and the other value outside the ring. Learning the optimal classifier would then amount to finding the appropriate ring.

Alternatively, one could augment the feature vector $\mathbf{x} = [x_1, x_2]^T$ with two new features equal to the squares of the existing ones, so that now

$$\mathbf{x} = [x_1, x_2, x_1^2, x_2^2]^T$$

or even *replace* the old feature values with their squares:

$$\mathbf{x} = [x_1^2, x_2^2]^T$$

or just their sum:

$$x = x_1^2 + x_2^2 \ .$$

Clearly, x is very informative, and the optimal classifier would identify an interval of x in which all red squares fall. If the other feature values are kept in $\mathbf{x}$, they become irrelevant.

We could even make the set linearly separable if we knew the radius r of the middle of the ring and the ring's half-width Δr: Just replace $\mathbf{x}$ by the single feature

$$x = \sqrt{|x_1^2 + x_2^2 - r^2|}$$

and then all points for which $x < \Delta r$ are in the ring, and are labeled `'s'`. All the other points are labeled `'c'`.

The key problem with all these drawings is that they are misleading in practice. Given a data set pertaining to some real classification or regression problem (think of SPAM filtering, or reading ZIP codes on envelopes), the data space X has typically a dimensionality d much higher than 2, and it is unclear what to draw as a 2D proxy for the real problem. Does the data distribution look more like in Figure 1 (a) or in Figure 1 (b)? Or does it have some special geometry as in Figure 1 (c)? We cannot just "look" at the data, because it lives in a space we cannot visualize. We don't even know if the training set is linearly separable, short of actually looking for a linear separator and then checking if we fail or succeed to find one.

Thus, while drawings are pedagogically useful to define concepts in machine learning, and heuristically useful when thinking of a machine learning problem, they must be used with care: It is easy to build some assumption into a drawing unwittingly, and then draw conclusions that fail if that assumption is not met.

Another danger in relying on drawings too heavily is that our intuition does not generalize well when thinking about spaces with more dimensions. For instance, we may talk about distances between data points, and imagine clusters of points or probability densities that summarize point distributions. Perhaps we draw a grid on the plane as a way to "discretize" a continuous problem, by saying that all points in the same cell of the grid are to be treated the same. All these concepts— points, curves, distance, probability distributions, clusters, grids— generalize to more dimensions, so we use similar abstractions as we think, say, of images, in which a single data point $\mathbf{x}$ has as many dimensions as there are pixels in an image, often millions, rather than just two or three dimensions.

Alas, this extension, from 2D to more dimensions, is fraught with danger in two major ways. The first is computational, and was named the *curse of dimensionality* by Richard Bellman in 1961.

What is simple on the plane may be practically impossible even in just 100 dimensions. As we saw, building a grid with a mere 10 cell boundaries per dimension requires 100 cells on the plane, but would require 10^{100} in a space with 100 dimensions. Since the number of atoms in the universe is estimated to be between 10^{78} and 10^{82}, working with a grid in 100 dimensions is clearly impossible. Similar difficulties arise when estimating probability densities or enumerating class boundaries.

The second danger is conceptual: Our intuitions about the geometry of data sets, function graphs, or class boundaries are based on our 2D or 3D experience. Alas, these intuitions are often wrong in spaces of many more dimensions. Consider for instance a circle inscribed in a square. Almost 79 percent of the area of the square is covered by the circle: $\pi r^2/(2r)^2 \approx 0.79$, and the distance from the center of the circle to a corner of the square is $\sqrt{2}r$, about 1.4 times the circle's radius. Thus, our intuition tells us that the circle covers most of the square, and that the corners of the square are less than two radius lengths away from the center. The remaining parts, between square and circle, are relatively unimportant (just 21 percent of the area). As we think of increasing dimensionality (a sphere inscribed in a cube, and so on), however, the volume in the sphere vanishes towards zero as a fraction of the volume of the cube, and the distance from the center to a corner of the cube goes to infinity as a multiple of the sphere radius. Thus, a d-dimensional sphere inscribed in a d-dimensional cube looks more like an octopus for large d: A vanishingly small sphere in the middle, with exponentially many, extremely long tentacles that extend towards the corners and contain almost all of the cube's volume!

A similar failure of our imagination in many dimensions relates to point distributions in space. When we think of, say, an isotropic Gaussian distribution of points around the origin of the plane and with standard deviation σ we think of a cloud of points that is densest at the origin and becomes sparser as we move further away from it. Thus, most of the points "fill" any circle around the origin whose radius is not much bigger than σ. This is not so in many dimensions. Even in just $d = 20$ dimensions, about 99 percent of all the points in the cloud are concentrated in a spherical shell centered at the origin, and whose inner radius is 0.8σ and whose outer radius is σ. As d increases further, the thickness of the shell vanishes: In many dimensions, essentially *all* points are on the surface of the sphere that is σ units away from the origin! Clearly, distances and distributions behave differently in many dimensions than they do in few.