Singapore University of Technology and Design
Information Systems Technology and Design
50.007 Machine Learning

Introduction

Much of what we do in engineering and sciences involves prediction. We can predict the weather tomorrow, properties of new materials, what people will purchase, and so on. Many of these predictions are data driven, i.e., based on past observations or measurements. For example, we can look at past weather patterns, measure properties of materials, record which products different people have bought/viewed before, and so on. In this course, we will look at such prediction problems as machine learning problems where the predictors are learned from data. We will try to understand how machine learning problems are formulated, which methods work when, what type of assumptions are needed, and what guarantees we might be able to provide for different methods. We will start with the simplest type of prediction problem – classification. In this case, the goal is to learn to classify examples such as biological samples, images, or text. Let's begin with a few scenarios.

- *Tumor classification.* Most tumors can be identified from tissue samples. While the samples themselves are easy to acquire from willing participants, the task of determining whether a sample contains tumor cells often involves a combination of laboratory tests and physician assessment. In order to easily screen large numbers of samples, we should develop automated methods that are capable of predicting a tumor/normal label for any new tissue sample. In machine learning, we call this problem a *classification problem.* Given some tissue samples with verified tumor/normal labels as *training data*, the goal is to find a good mapping from samples to labels – a *classifier.* Once found, the classifier can be easily applied to predict labels for new samples.

  But computers do not understand "tissue samples". We have to work a bit harder to describe each sample in a manner that can be used by automated methods. We could, for example, try to represent each sample with a *feature vector.* To construct such a vector, we could use readily available gene expression assays to first measure how active each gene is in a population of cells represented by the sample. In this case, each coordinate of the feature vector would correspond to the resulting expression of a particular gene. The assumption here is that the expression of genes would provide sufficient information to determine whether the sample is a tumor. Once feature vectors are available, we can apply a generic machine learning method to learn a mapping from vectors to labels. Such a method would be completely oblivious to how the vectors were constructed, and what the coordinates mean. All it would do is relate different coordinate values (or their combinations) to the corresponding labels in the training data.

It is important that we follow the same protocol to construct feature vectors for training samples and any new samples to be classified. Otherwise the new samples would "look" different to the classifier and result in poor predictions. There are many subtleties here that we will address more formally later on in the course. For example, if the classifier is trained on samples from one type of tumor but tested on samples taken from another type of tumor, there's little reason to expect (in general) that the classifier would do well. We are making a tacit assumption that the samples used for training are somehow representative of the samples that the classifier sees (and tries to classify) later on.

- *Gender from images.* Most people would find it relatively easy to determine the gender of a person based on portraits. But a human-powered solution is expensive, and we would rather use an automated method for predicting gender across large sets of images. In order to set up gender classification as a machine learning problem, we will first need a bit of human assistance to create a set of labeled images, i.e., (image,gender) pairs. These labeled faces constitute the training set for the classification method. As in the tissue classification problem, the learning task is to relate the descriptions of images (as feature vectors) to the corresponding gender labels. Once the mapping from images to labels is found, we can easily apply the mapping to label large numbers of new images.

  How do we represent images as feature vectors? We could take a high resolution pixel image of a face and simply concatenate all the pixel values (color, intensity) into a long feature vector. While possible, this may not work very well. The reason is that we are leaving everything for the classification method to figure out. For example, hair, skin color, eyes, etc may be important "features" to pay attention to in order to predict gender but none of these features are deducible from individual pixels. Indeed, images in computer vision are often mapped to feature vectors with the help of simple detectors that act like classifiers themselves. They may detect edges, color patches, different textures, and so on. It is more useful to concatenate the outputs of these detectors into a feature vector and use such vectors to predict gender. More generally, it is important to *represent* the examples to be classified in a way that the information pertaining to the labels is more easily accessible.

Classification as machine learning

Let's look at these types of classification problems a bit more formally. We will use $x = [x_1, \ldots, x_d]^T \in \mathcal{R}^d$ to denote each feature (column) vector of dimension d. To avoid confusion, when x is used to denote the original object (e.g., sample, image, document), we will use $\phi(x) \in \mathcal{R}^d$ for the feature vector constructed from object x. But, for now, let's simply use x as a column feature vector, as if it was given to us directly. In other words, let's look at the problem as the classification method sees it.

Each training example x is associated with a binary label $y \in \{-1, 1\}$. For new examples, we will have to predict the label. Let's say that we have n training examples available to learn from. We will index the training examples with superscripts, $x^{(1)}, x^{(2)}, \ldots, x^{(n)}$ and similarly for the corresponding labels $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$. All that the classification method knows about the problem is the training data as n pairs $(x^{(i)}, y^{(i)})$, $i = 1, \ldots, n$. Let's call this training data S_n where the subscript n highlights the number of examples we have.

A classifier h is a mapping from feature vectors to labels: $h : \mathcal{R}^d \to \{-1, 1\}$. When we apply the classifier to a particular example x, we write $h(x)$ as the predicted label. Any learning algorithm entertains a set of alternative classifiers $\mathcal{H}$, and then selects one $\hat{h} \in \mathcal{H}$ based on the training set S_n. The goal is to select $\hat{h} \in \mathcal{H}$ that would have the best chance of correctly classifying new examples that were not part of the training set. Note the difficulty here. All the information we have about the classification problem is the training set S_n but we are actually interested in doing well on examples that were not part of the training set. In other words, we are interested in prediction.

Our brief discussion is already touching on some of the key aspects of learning problems:

1. *Set of classifiers* $\mathcal{H}$. Depending on the context, this set is also known as *the model* or *the hypothesis class*. The larger this set is, the more powerful the classification method is. In other words, there are many classifiers to choose from in response to the training set, many alternative hypotheses about how feature vectors relate to labels. We will later formalize exactly how to think about the size of $\mathcal{H}$. For now, to gain intuition, you can think of it as a discrete set, where each member is a (slightly) different classifier.

2. *Learning algorithm/criterion*. The problem of finding $\hat{h} \in \mathcal{H}$ based on the training set S_n is solved by the learning algorithm. This training problem is often cast as an optimization problem, and we will talk about the *objective function or estimation criterion* for selecting $\hat{h}$ from $\mathcal{H}$. Note that many learning algorithms or corresponding estimation criteria might use the same set of classifiers $\mathcal{H}$ but still select different classifiers in response to the training set.

3. *Generalization*. The goal of the learning algorithm is to find a classifier $\hat{h} \in \mathcal{H}$ that will work well on yet unseen examples x. Put another way, we say that we want a classifier that *generalizes* well. How well the resulting classifier will work on new examples depends on the choice of $\mathcal{H}$, the training data S_n, as well as the learning algorithm. A classifier that predicts all the training labels correctly may not generalize well. In order to generalize, we must capture something about how the feature vectors relate to the labels.

Let's take a simple example to see how these concepts relate, and what we must do to generalize well. Consider the gender classification task based on images discussed above.

Assume that each image (grayscale) is represented as a column vector x of dimension d. The pixel intensity values in the image, column by column, are concatenated into a single column vector. If the image has 128 by 128 pixels, then $d = 16\,384$. We assume that all the images are of the same size. You may remember that we had already argued against using this simple vector representation for images. Indeed, there are many better ones. However, we will use this easy-to-understand representation to illustrate some of the basic learning issues.

A classifier in this context is a binary valued function $h : \mathcal{R}^d \to \{-1, 1\}$ chosen on the basis of the training set alone. For our task here we assume that the classifier knows nothing about images (or faces for that matter) beyond the labeled training set. So, for example, from the point of view of the classifier, the images could have been measurements of weight, height, etc. rather than pixel intensities. The classifier only has a set of n training vectors $x^{(1)}, \ldots, x^{(n)}$ with binary ±1 labels $y^{(1)}, \ldots, y^{(n)}$. This is the only information about the task that we can use to constrain what the classifier $\hat{h}$ should be.

In order to learn anything, we will have to constrain the set of classifiers $\mathcal{H}$. Put another way, effective learning requires constraints. Let's see first what would happen without any constraints. To this end, suppose we have $n = 50$ labeled 128×128 pixel images where the pixel intensities range from 0 to 255. Given the small number of training examples, it is likely that one of the pixels, say pixel i, has a distinct value in each of the training images. If $\mathcal{H}$ includes all possible classifiers (no constraints), then we could also find a classifier that relies on the value of this single pixel alone, yet perfectly maps all the training images to their correct labels. Let $x_i^{(t)}$ refer to pixel i in the t^{th} training image, and suppose x'_i is the i^{th} pixel in any image x'. Then our simple single pixel classifier could be written as

$$\hat{h}(x') = \begin{cases} y^{(t)}, \text{ if } x_i^{(t)} = x'_i \text{ for some } t = 1, \ldots, n \text{ (in this order)} \\ -1, \text{ otherwise} \end{cases} \tag{1}$$

You should verify that this classifier does indeed map training examples to their correct labels. In fact, when there are no constraints on $\mathcal{H}$, it is always possible to come up with such a "perfect" classifier if the training images are all distinct (no two images have identical pixel intensities for all the pixels). Any training algorithm that tries to find classifiers that make few errors on the training set could end up selecting such $\hat{h}$.

But we are forgetting here that the task is *not* to correctly classify the training images; the training set is merely a helpful source of information. Our task is to do well on yet unseen images. Do we expect our single pixel classifier to correctly classify images not in the training set? New images are likely to portray different people, in different orientations, under varying lighting conditions, etc. The value of the single pixel (e.g., background) is likely to bear no relevance to the gender label.

What went wrong? Our set $\mathcal{H}$ (no constraints) is too large. It contains classifiers which do well on the training set but perform poorly on new images. This is a subtle but important sentence. Any fixed classifier, however complicated it may be, would statistically speaking do about the same on the training set as on new images. The problem here is not a particular classifier but the fact that there are so many choices available in $\mathcal{H}$ that we may end up choosing the one that does do substantially better on the specific training set we have than on yet unseen images. We say that in such cases $\mathcal{H}$, i.e., the set of classifiers, *overfits* the training data. The training set is too small in relation to $\mathcal{H}$ in order for us to have any statistical power to distinguish between all the available choices $h \in \mathcal{H}$.

In order to find classifiers that *generalize* well, we must constrain the set $\mathcal{H}$. We would like to find a set of classifiers such that if a classifier chosen from this set works well on the training set, it is also likely to work well on the unseen images. This right set of classifiers cannot be too large in the sense of containing too many clearly different functions. Otherwise we are likely to find classifiers such as the trivial ones that are close to perfect on the training set but do not generalize well. The set of classifiers should not be too small either or we run the risk of not finding any classifiers that work well even on the training set. For example, suppose $\mathcal{H}$ contains only one classifier. There's really no learning here, we will choose the same $h \in \mathcal{H}$ regardless of the training set. But, we do know that how well it does on the training set is probably a good measure of how well it does on new images. Finding the right set is a key problem in machine learning, also known as the *model selection* problem. We will discuss it more later.

Linear classification

Let's start by considering a particular constrained set of classifiers. Specifically, we will look at *linear classifiers*. These are thresholded linear mappings from images to labels. More formally, we only consider classifiers of the form

$$h(x;\theta) = \text{sign}\big(\,\theta_1 x_1 + \ldots + \theta_d x_d\,\big) = \text{sign}\big(\,\theta \cdot x\,\big) = \begin{cases} +1, & \theta \cdot x \geq 0 \\ -1, & \theta \cdot x < 0 \end{cases} \tag{2}$$

where $\theta \cdot x = \theta^T x$ and $\theta = [\theta_1, \ldots, \theta_d]^T$ is a column vector of real valued parameters. Different settings of the parameters give rise to different classifiers in this set. In other words, θ indexes the classifiers. Any two classifiers corresponding to different parameters would produce a different prediction for some input images x. We say that the classifiers in this set are *parameterized* by $\theta \in \mathcal{R}^d$.

We can also understand these linear classifiers geometrically. Suppose we select one classifier, i.e., fix parameters θ, and look at what happens for different images x. The classifier changes its prediction only when the argument to the sign function changes from positive to negative (or vice versa). Geometrically, in the space of image vectors, this transition corresponds to crossing the *decision boundary* where the argument is exactly zero: all images x such that $\theta \cdot x = 0$ lie exactly on the decision boundary. This is a linear equation in x (θ is fixed). It defines a plane in d-dimensions, a plane that

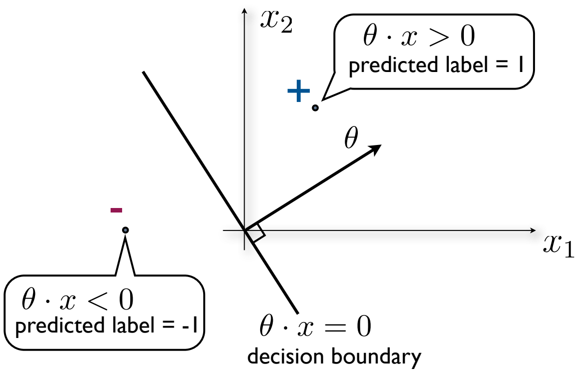

Figure 1: A linear classifier through origin.

goes through the origin $x = 0$ (since $\theta \cdot 0 = 0$). What is the direction of this plane? The parameter vector θ is normal (orthogonal) to this plane; this is clear since the plane is defined as all x for which $\theta \cdot x = 0$. The θ vector as the normal to the plane also specifies the direction in the image space along which the value of $\theta \cdot x$ would increase the most. Figure 1 tries to illustrate these concepts in two dimensions.