Machine Learning: A very quick introduction

Sumeet Agarwal

January 6, 2013

Machine learning [1] is concerned with algorithmically finding patterns and relationships in data, and using these to perform tasks such as classification and prediction in various domains. We now introduce some relevant terminology and provide an overview of a few sorts of machine learning approaches.

1 Basics

- *Feature vector*: A typical setting for machine learning is to be given a collection of objects (or data points), each of which is characterised by several different *features*. Features can be of different sorts: e.g., they might be continuous (say, real- or integer-valued) or categorical (for instance, a feature for colour can have values like `green,blue,red`). A vector containing all of the feature values for a given data point is called the *feature vector*; if this is a vector of length d, then one can think of each data point as being mapped to a d-dimensional vector space (in the case of real-valued features, this is $\mathbb{R}^d$), called the *feature space*.
- *Design matrix*: A collection of feature vectors for different data points constitutes a *design matrix*. Each row of the matrix is one data point (i.e., one feature vector), and each column represents the values of a given feature across all of the data points (Table 1). The design matrix is the basic data object on which machine learning algorithms operate.

2 Supervised learning

The task of supervised learning is to learn an association between features and external labels of some kind. A label is typically either one of a finite set of categories (in which case it becomes a classification problem), or continuous-valued (in which case one has a regression problem). We discuss both of these settings next.

2.1 Classification

Given a set of objects represented as feature vectors and an associated class label for each object, one would like to learn a model (known as a *classifier*) that can predict the class given the features. The model itself can take on many different forms: linear classifiers, decision trees, neural networks, and support vector machines are a few popular examples [1]. Here we will briefly discuss the first two.

Table 1: **Example design matrix.**

Object	Weight (g)	Colour (0=Green, 1=Red)
Red Apple 1	147	0.90
Red Apple 2	159	0.70
Red Apple 3	170	0.77
Green Apple 1	163	0.17
Green Apple 2	151	0.13
Banana 1	104	0.10
Banana 2	119	0.15
Banana 3	113	0.34
Banana 4	122	0.23
Banana 5	125	0.30

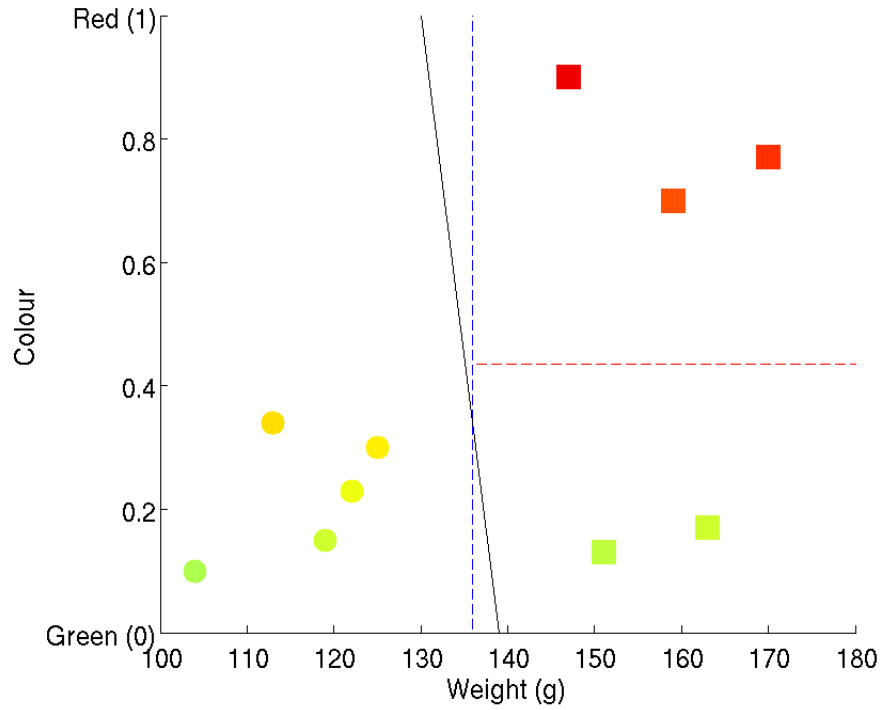
Design matrix for 10 objects and 2 numerical features. The colour spectrum from green to red is mapped to a 0–1 scale (see also Figure 1).

A *linear classifier* uses some linear combination of the features as its criterion for distinguishing between classes [1, 3]. This corresponds to drawing a separating hyperplane in the feature space; in two dimensions, this is a line, as in Figure 1(a). Thus, linear classifiers by default are defined for binary classification problems—i.e., those in which there are only two classes. When there are more than two classes, it is typical to use multiple linear classifiers; two possible approaches are *all-vs-all*, in which a binary classifier is learnt for every pair of classes, and *one-vs-all*, in which a binary classifier is learnt to discriminate each class from the combination of all of the other classes.

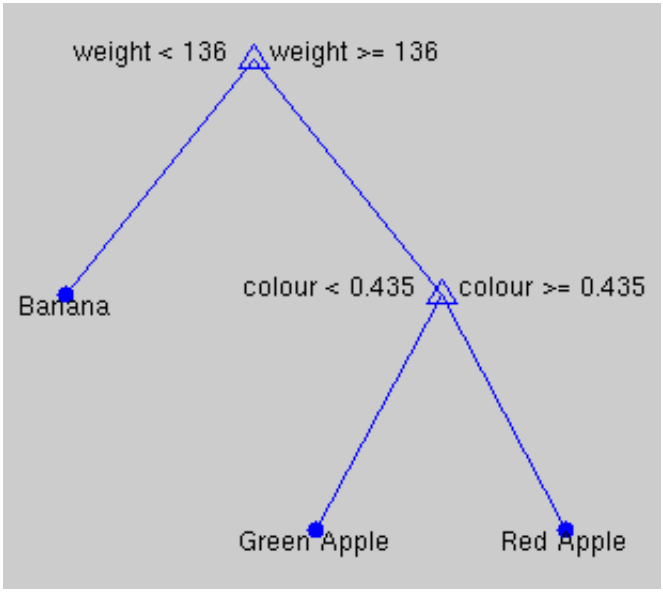
Decision trees consist of a set of rules based on feature values [2]; they are arranged in the form of a binary tree, as in Figure 1(b). Following these rules down the tree specifies increasingly restricted regions of feature space until at some point a leaf node is reached and all points in the corresponding region get assigned to a single class.

Having chosen a particular form of model, one then needs to use the data to learn a specific classifier—typically one that optimises some performance measure. An obvious choice for this measure might be what fraction of the given data points the classifier is able to place in the correct class (known as classification *accuracy*). However, this suffers from the problem of *overfitting*, in that one would like to use the classifier to make predictions on novel data points, and the data set at hand will in general not be representative of the full underlying distribution of points in feature space [1,3]. Thus, the learnt classifier might tend to fit peculiarities of the data set, thereby worsening its performance on unseen examples. In order to avoid this, it is usual to evaluate a classifier not on the data set used to learn it (called the *training set*), but rather on a separate set (the *test set*); this is known as *out-of-sample* evaluation [1, 3]. Some fraction of the available data (10% or 20% are typical choices) would be designated as the test set and would not be used to train the classifier (but instead to evaluate it). One popular variant of this approach, which allows the use of all data for training, is known as *cross-validation* [7]. In this approach, the data is split into k equal parts, known as *folds* (a common choice is $k = 10$, in which case it is called 10-fold cross-validation).[1] Subsequently, k different classifiers are trained—each time with one of the k

[1]Larger values of k lead to more robust error estimates, as a larger number of classifiers are averaged over and each classifier is trained on a larger number of data points. Thus in this sense the optimal value for k is equal



(a)



(b)

Figure 1: **Example classifiers.**

(a) Data points from Table 1 in feature space (colour represented both visually and numerically on the y-axis): one can split them into two classes, bananas (circles) and apples (squares). The black line is a linear classifier separating the data. The apples can be further split into red and green varieties; the dashed lines show the partitions imposed by a decision-tree classifier for the three-class problem. (b) The decision tree.

parts used as the test set and the rest used as the training set. Thus, the combined test results of these k classifiers allow one to estimate out-of-sample accuracy on the entire data set. This can then be used as a criterion for classifier choice—for instance, via setting model parameters.

2.2 Regression

When the dependent variable—i.e., the one we would like to predict, given the features we have for the data points—is not a categorical class label but instead a continuous (typically real-valued) quantity, then learning a predictive model for this can be seen as a regression problem. One is required to find a function f that maps from a feature vector $\mathbf{x}$ to an output y: ideally, $y = f(\mathbf{x})$. The simplest form is *linear regression*, analogous to linear classification, in which f is a linear combination of the features (plus a possible constant term or offset): $f(\mathbf{x}) = \mathbf{w}.\mathbf{x} + b$ [1,3]. Here $\mathbf{w}$ is a *weight vector* that represents the coefficients of the different features in f. Thus, $\mathbf{w}$ and b are the parameters in a linear regression model that are to be learnt from the data.

The concepts of training, testing, and cross-validation can be extended to regression once one has defined an appropriate accuracy measure. Suppose one is given a data set $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_r, y_r)$, where r is the number of data points. We use this to learn a regression function f, such that $\hat{y}_i = f(\mathbf{x}_i)$. Thus, $\hat{y}_i$ would be the predicted value of the dependent variable for the i^{th} data point. The difference $y_i - \hat{y}_i$ is known as the *residual* for the i^{th} point; this is a measure of the error the learnt function makes in predicting for this instance. In general, lower residuals correspond to higher accuracy; the most common way of evaluating accuracy over a set of points is to take the sum of the squares of the residuals: $\sum_{i=1}^{r}(y_i - \hat{y}_i)^2$. This is sometimes known as the *deviance* [6] or the *squared error* [1,3].

3 Unsupervised learning

The task of unsupervised learning is to find patterns in data without any external labelling; most commonly, the patterns of interest are clusters, in which case it is also described as *clustering*. There are a large number of approaches for unsupervised learning [1,3]; we will now discuss a few.

Single-linkage clustering is a distance-based method that involves initially defining a distance measure between pairs of points [11]. If the points lie in a vector space, as in Figure 1(a), then this can be a standard measure like Euclidean distance. Having computed the distance between every pair of points, this method then proceeds by initially assigning each point to a separate cluster, and then iteratively finding and merging the closest pair of clusters until all of the points have been lumped into a single cluster. The distance between a pair of clusters is defined as the minimum of all pairwise distances between points across the two clusters.[2] This leads to a hierarchical clustering: at each iteration of the process, one moves up the hierarchy. One can use a threshold

to the size of the data set, which corresponds to having just one point in the test set each time; this is known as *leave-one-out* cross-validation. However, larger values of k also mean greater computational cost in re-running the training algorithm for each fold; thus in practice relatively small values of k are preferred, with $k = 5$ or $k = 10$ being widely used choices.

[2]Other common choices for this distance measure include the average of all pairwise distances, which leads to *average-linkage* clustering, and the maximum of all pairwise distances, which leads to *complete-linkage* clustering. One drawback of using single-linkage clustering is that it may lead to clusters where some elements are very far apart, as clusters are merged based only on the distance between the closest elements. More generally, this sort of agglomerative clustering is a standard, simple method but the results may not always be easy to interpret, as it gives a hierarchy of clusters at different levels, rather than a single partition.

to specify the minimum distance between clusters to terminate the iteration at a particular point and obtain a single set of clusters.

In order to detect patterns in data, it is often useful to map it to a low-dimensional space, where the number of dimensions is typically chosen to be as low as possible whilst capturing the bulk of the variability in the given data set. The canonical way of doing this is via *principal component analysis* (PCA) [8]. The essential idea is to find directions in feature space along which the spread of the data is the greatest; each direction is given by some linear combination of the features. This can be done via *singular value decomposition* (SVD), which is, for non-square matrices, the analog of eigendecomposition [1, 9]. Suppose we denote the $r \times d$ design matrix by X. According to the SVD theorem, this can then be factorised as $X = V\Sigma W^T$, where V is an $r \times r$ orthogonal matrix of eigenvectors of XX^T, W is a $d \times d$ orthogonal matrix of eigenvectors of X^TX, and Σ is an $r \times d$ matrix with nonnegative numbers along the diagonal (with all other entries equal to 0). The PCA transformation is given by $Y = XW$; the matrix Y is also an $r \times d$ matrix; it represents the design matrix in the transformed feature space (the features of which are the principal components). These features will be in decreasing order of the amount of data variance they capture. In order to obtain a reduced representation (say, in l-dimensional space), one can take the first l columns of W. If we denote this by W_l, then the design matrix in the l dimensions is given by $Y_l = XW_l$. In practice, it is often useful to choose $l = 2$ in order to produce a two-dimensional plot of the data; this allows for visual inspection and can aid in the detection of intuitive clusters or patterns. However such dimensionality reduction of course also involves throwing away information, and one has to be cautious in interpreting the results, particularly if the reduced dimensions leave a substantial proportion of the variance in the data uncaptured.

One limitation of PCA is that the reduced dimensions must be linear combinations of the given features. It can sometimes be useful to select "directions" that are not straight lines in feature space; for instance, if all of the data points lie along a circle, then one actually needs only a single dimension to capture the variation between them, but PCA will not be able to detect this. To account for this, several methods have been developed in recent years for non-linear dimensionality reduction [4, 5, 10, 12]; the one we will discuss briefly is known as *Isomap* [12]. The idea behind Isomap is to capture the local geometry of the surface on which the points sit in feature space. A weighted graph using these points as nodes is defined as follows: each data point is connected to its k nearest Euclidean neighbours in the space with links of weight equal to the Euclidean distance, with the parameter k to be specified by the user.[3] A distance matrix D between points is then defined by using weighted distances in this graph. One obtains the eigendecomposition of D (which is analogous to X^TX above), and the top l eigenvectors (analogous to W_l) then define the coordinates for an l-dimensional embedding. The amount of data variability captured in the reduced space can be quantified via the *residual variance*, which can be computed as $1 - R^2(D, D_l)$, where R denotes the linear correlation coefficient, and D_l is the matrix of pairwise Euclidean distances between points in the l-dimensional embedding.

References

[1] C. Bishop. *Pattern Recognition and Machine Learning*. Information science and statistics. Springer (2006).

[3] One would like k to be relatively small, as the objective is to approximate local geometry. However, if k is too small then it might lead to a sparse or disconnected graph. In our usage we choose k to be the smallest number which leads to a connected graph containing all the data points in the set under consideration.

[2] L. Breiman. *Classification and regression trees*. The Wadsworth and Brooks-Cole statistics-probability series. Chapman & Hall (1984).

[3] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer (2009).

[4] N. Lawrence. Probabilistic non-linear principal component analysis with gaussian process latent variable models. *Journal of Machine Learning Research*, 6:1783–1816 (2005).

[5] J. A. Lee and M. Verleysen. *Nonlinear Dimensionality Reduction*. Springer (2007).

[6] E. P. Martins. Estimating the rate of phenotypic evolution from comparative data. *The American Naturalist*, 144(2):193–209 (1994).

[7] F. Mosteller and J. W. Tukey. Data analysis, including statistics. In *Handbook of Social Psychology*. Addison-Wesley, Reading, MA, USA (1968).

[8] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572 (1901).

[9] W. Press. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press (2007).

[10] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326 (2000).

[11] R. Sibson. SLINK: an optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34 (1973).

[12] J. B. Tenenbaum, V. Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323 (2000).