Lecture Notes: Introduction to ML Concepts

Ted Meeds[1,2]

[1] Informatics Institute, University of Amsterdam
[2] The Centre for Integrative Bioinformatics, Vrije University
tmeeds@gmail.com

Abstract In this lecture note, the very basic concepts of Machine Learning are covered, including what is machine learning, what are its goals, etc. The core concept of data sets, its representation, and some examples are given. Concepts of model, objective function, algorithm, parameters, generalization, overfitting are introduced using linear regression as an example.

Keywords: Machine Learning concepts, Data set representations

1 Introduction

Pattern recognition is a general term meaning the automatic discovery of regularities in data, but it is synonymous with *machine learning* which is now the accepted name for the field. In machine learning (ML), *algorithms* are implemented in computer programs for the training or *learning* models of data. Models are then used for *prediction* of new, unseen data (i. e., at *test-time*). The primary goal of learning is to uncover *regularities* in the data, captured by the model parameters, that are exploited to take actions; e. g., predict the class of an image, make a diagnosis (a decision) based on the predicted probabilities of a disease, etc.

2 Representing Data

Data is a fundamental object of interest in ML. It is important to understand the shorthand used to represent data cases and data sets so that reading and communicating ML equations, models, algorithms, is easy and consistent.

2.1 Data vectors

A single data case may be expressed as a data vector

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{bmatrix} \tag{1}$$

is a column vector of D scalars x_d, where d is an index with values from 1 to D. When writing by hand, vectors can be written with an underscore, $\underline{\mathbf{x}}$, or with an arrow, $\overrightarrow{\mathbf{x}}$. My preference is $\underline{\mathbf{x}}$ because it extends without much clutter to multi-dimensional matrices, but either is fine as long as it is consistent. We write the *transpose* of $\mathbf{x}$ as $\mathbf{x}^T = \begin{bmatrix} x_1, x_2, \cdots, x_D \end{bmatrix}$. Note that a data cases/examples may come in pairs, or tuples of data vectors (see below).

2.2 Data matrices

A data set may be expressed as a data matrix

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,N} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \cdots & x_{N,D} \end{bmatrix} \tag{2}$$

is an N by D matrix with scalar entries $x_{n,d}$. We typically drop the ",":

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1N} \\ x_{21} & x_{22} & \cdots & x_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \tag{3}$$

Note that $\mathbf{X}$ has N transposed data vectors stacked vertically:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} \tag{4}$$

As with data cases, data sets may come in pairs/tuples of data matrices $(\mathbf{x}_n, \mathbf{t}_n)$, typically called input-output pairs.

2.3 Input-output data

Data sets may come in pairs of data matrices $\{\mathbf{X}, \mathbf{T}\}$. Typically $\mathbf{X}$ represents *input* data, such as images in an image classification problem, or other measurements that are *observed* at test-time. Output data matrices $\mathbf{T}$ are only observed during *training* and are unobserved at test-time. $\mathbf{T}$ represents *targets* ("t" for target; i. e., predict the target t_n using the input vector $\mathbf{x}_n$).

$$\mathbf{t}_n = \begin{bmatrix} t_{n1} \\ t_{n2} \\ \vdots \\ t_{nK} \end{bmatrix} \qquad \mathbf{T} = \begin{bmatrix} \mathbf{t}_1^T \\ \mathbf{t}_2^T \\ \vdots \\ \mathbf{t}_N^T \end{bmatrix} \tag{5}$$

For many problems we encounter, the number of outputs $K = 1$, for these cases we drop the second index over K and write

$$\mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{bmatrix} \tag{6}$$

(i.e. $\mathbf{t}$ is used instead of $\mathbf{T}$.) In other literature, the letter y is often used (the y-axis values are a function of the x-axis values). In Bishop, y will usually be a prediction (estimate) our model makes about t. A data set can therefore be written as a pair: $(\mathbf{X}, \mathbf{t})$ (when $K = 1$) or $(\mathbf{X}, \mathbf{T})$ (when $K > 1$). The "data" is sometimes written as $\mathcal{D}$, for short. Remember, data means observations. This will come up when we discuss *likelihood* functions (i. e., how likely are our observations given a set parameter values).

2.4 More notation

- N: the number of images in the collection / data set.
- D: the number of pixels in an image (and we assume this is the same for each image).
- K: the number of **labels** or **tags** in the collection.
- $\mathbf{x}_n$: a vector of pixel values taken row-wise from the n^{th} image.
- $\mathbf{t}_n$: a vector of zeros except for the label index, e.g. $t_{nk} = 1$ (if label is k).

2.5 Example: MNIST data

Perhaps the most well-known (and therefore overused) datasets in ML is the MNIST classification dataset. The data consist of handwritten digits (the inputs) that are labelled 0-9 (the outputs/targets). The MNIST data set are images of handwritten digits that have been manually labelled (so $K = 10$). The images are of size 28 by 28 (so $D = 28 * 28 = 784$). There are two datasets: the **training** dataset where $N = 60000$ and the **test** dataset where $N = 10000$. We may want distinguish the two sizes when reporting on experiments: N_{train} v N_{test} (and $\mathcal{D}_{train}$ v $\mathcal{D}_{test}$, etc.) The training set is used to modify the parameters of a model to optimize some criteria and the test set is used to evaluate the performance on **unseen** data.

More info about MNIST:

- N: the number of images in the collection / data set.
- D: the number of pixels in an image (and we assume this is the same for each image).
- K: the number of **labels** or **tags** in the collection.
- $\mathbf{x}_n$: a vector of pixel values taken row-wise from the n^{th} image.
- $\mathbf{t}_n$: a vector of zeros except for the label index, e.g. $t_{nk} = 1$ (if label is k).

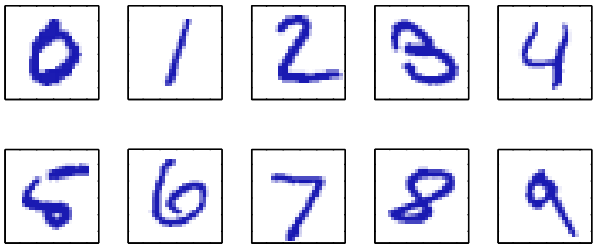

Figure 1. MNIST data set (Bishop 1.1)

3 Models, Algorithms, etc

Based on the desired actions and type of data, a model (or model class) is assumed. A model describes our **inductive bias**: the set of assumptions we make about the relationship between input and output. An action may be predicting a continuous valued target (regression). MNIST is a classification problem since the desired action is to predict a discrete label. We may also want to predict the probability of a label, and make decisions based on the predicted probabilities and a set of costs for each decision. A model has (vector) parameters $\boldsymbol{\theta}$. We have written the parameters as a vector, but they can be scalars, vectors, matrices, and mixtures of these and be of different data types).

An algorithm $\boldsymbol{\theta}$ by optimizing (minimizing/maximizing) an **objective function** that is a function of $\boldsymbol{\theta}$ and, implicitly $\mathcal{D}$ and the model class. A model class may admit $\boldsymbol{\theta}$ of varying complexity (e.g. M polynomial basis functions). More complex models have more **capacity** to capture more structure from the data.

3.1 Example: curve fitting (regression) with polynomial basis functions.

Model class and complexity The model class is linear regression using polynomial basis functions of degree M.

Parameters A **weight** vector $\mathbf{w}$ (this is $\boldsymbol{\theta}$, but using 'w' is more intuitive.)

Objective function sum-of-squared errors

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(y\left(x_n, \mathbf{w}\right) - t_n\right)^2 \tag{7}$$

The model is a polynomial function $y\left(x_n, \mathbf{w}\right)$ that implicitly depends on M; i. e.,

$$y\left(x_n, \mathbf{w}\right) = w_0 + w_1 x + w_1 x^2 + \cdots + w_{M-1}x^{M-1} = \sum_{m=0}^{M-1} w_m x^m \tag{8}$$

Note here we assume the data have $D = 1$, hence the scalar x_n instead of the vector $\mathbf{x}$. Note sometimes we might write y_n for $y(x_n, \mathbf{w})$. Later we will introduce the idea of basis functions and use $\boldsymbol{\phi}(\mathbf{x}_n)$ or $\boldsymbol{\phi}_n$ for short, to represent the vector of basis functions. For example,

$$\sum_{m=0}^{M-1} w_m x^m = \boldsymbol{\phi}(\mathbf{x}_n)^T \mathbf{w} \tag{9}$$

This is why we call this linear regression: it is a linear function of the basis vector (even if in the input space it is a non-linear function).

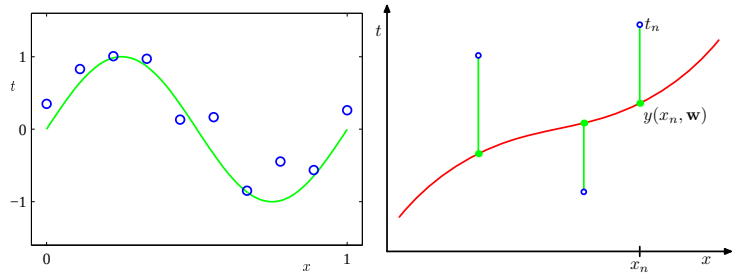

Figure 2. Linear regression (Bishop 1.2, Bishop 1.3. Left: The true generating function in green with data in blue circles. Right: A few data vectors from the training set and the estimated function y. Note the distance from the targets t_n and $y(x_n, \mathbf{w})$. During training, $\mathbf{w}$ is adjusted by minimizing the sum of squared distances between the targets and the estimator.

Spring Analogy We can use a simple spring analogy to understand the squared loss in regression. Recall that the force in a spring is $f = -kx$, where k is the spring constant and x is the distance the spring is stretched. The potential energy in the spring is $\frac{1}{2}kx^2$. We can therefore think of squared loss cost function as the sum of potential energies across several springs, i. e., individual data points. Each data point exerts a force proportional to the distance of the estimate to the observation.

Training the model Algorithm (for adjusting $\mathbf{w}$): least-squares (we will see later). Action: predict t at locations x; our prediction is y. Note our action assumes we do not know t at test-time, which implies that we are interested in generalized prediction (on examples we haven't seen in our training set). During the training procedure, we can test the model (use its predictions) on "held-out" data; the prediction error would improve as the model learns, but will overfit if the model is too complex. This means it has learned more than the inherent

structure in the data and has begun learning the noise. This leads to the concept of **generalization**, a fundamental concept in ML. Generalization means that the model performs well on unseen data. When a model performs poorly on unseen data, but well on training data, we say that it has **overfit**. We devote a lot of effort in the detection, avoidance of **overfitting** using **regularization** and **model selection**.

Probabilistic perspective A probabilistic perspective is often used in ML (and we can sometimes view non-probabilistic objectives as probabilistic, e.g. the squared-loss implies a Gaussian conditional distribution).

More info:

- $p(t_n|x_n)$ is the **predictive distribution** of target t_n given input x_n. Probability provides the tools for handling uncertainty we have because we have a finite data set and noisy measurements. It will also help model uncertainty.
- $y(x_n, \mathbf{w}) = \mathbb{E}[t_n|x_n] = \int t_n p(t_n|x_n) dt_n$. i. e., our prediction is the expected value under our predictive distribution. Note that this is the optimal decision if we have a squared loss (more on this later when we discuss loss functions for regression).
- If the distribution $p(t_n|x_n)$ is **Gaussian**, then we have $E(\mathbf{w}) = -loglikelihood$. i. e., there is an equivalence between using squared loss and a Gaussian log-likelihood when we take a probabilistic interpretation.

References