Machine Learning

Sina Aleyaasin

November 7, 2017

1 Preliminaries

Computational approaches to *learning* from data involves some or all of following processes.

1. **Feature engineering** encoding information about a learning problem into machine processable format.
2. **Model selection** employing appropriate models to describe data
3. **Training** process existing empirical data to make hypotheses that predict future, unknown data points.

Techniques in machine learning coarsely divide into two categories

- Supervised learning
- Unsupervised learning

1.1 Feature engineering

Consider the problem of classifying objects from a set of examples $\chi \in X$. To represent the problem in the domain of a machine, we must represent the objects using numbers.

Definition 1.1. A **feature vector** is some vector representation of an object x

$$x \equiv \phi(\chi) \in \mathbb{R}^d \tag{1}$$

The exercise of selecting ϕ is referred to as *feature enginering.* We may express the set of examples X in this way as a matrix.

Definition 1.2. A **design matrix** is a matrix of N feature vectors representing N objects

$$X = \begin{bmatrix} x^{(1)T} \\ x^{(2)T} \\ \vdots \\ x^{(N)T} \end{bmatrix} \tag{2}$$

1.2 Supervised Learning

Consider the task of modelling the relationship between some dependent variable (*target*) on some explanatory independent variable (*feature*) given a data set.

Definition 1.3. A **training example** is an ordered *feature-target* pair

$$(x^{(i)}, y^{(i)}) \in \mathbb{X} \times \mathbb{Y} \tag{3}$$

Definition 1.4. A **training set** of size N is the set of training examples

$$\{(x^{(i)}, y^{(i)}) | i = 1, 2, \dots N\} \tag{4}$$

The objective is to learn a *hypothesis*

$$h : \mathbb{X} \to \mathbb{Y} \tag{5}$$

that models the relationship between the data in the training set and can predict new data points beyond it.

Definition 1.5. **Regression** is supervised learning for a continuous, real valued $\mathbb{Y} \equiv \mathbb{R}^{n+1}$.

Definition 1.6. **Classification** is supervised learning for a finite, discrete $\mathbb{Y}$. The hypothesis of a classification problem is also known as a **classifier**.

1.3 Model selection

$\mathcal{H}$

1.4 Training

$h \in \mathcal{H}$

2 Linear Regression

Linear regression assumes a linear dependence of the target on the features.

$$h(x; \theta) \equiv h_\theta(x) = \theta^T x \tag{6}$$

where $\theta, x \in \mathbb{R}^{n+1}$.

Remark. Motivated by aesthetics, this notation adheres to the convention that every feature vector x has a constant first element $x_0 = 1$ to account for the intercept term θ_0.

The parameter θ is determined by minimizing some *loss function* that aims to quantify the "error" of the classifier. We consider here the loss function giving corresponding to the **ordinary least squares** regression model

$$J(\theta) = \sum_{i=1}^{N} J^{(i)}(\theta) = \sum_{i=1}^{N} \frac{1}{2}\left(y^{(i)} - h_\theta(x^{(i)})\right)^2 \tag{7}$$

2.1 Minimizing loss by normal equations

In some instances, as with the least mean squares regression model, the loss function may be minimized analytically to yield a closed formed solution for θ

theorem and proof of linear algebra + calculate theta

2.2 Minimizing loss by gradient descent

For instances where no closed form solutions exist, one may perform a **gradient descent** until a desired threshold of convergence is reached.

$$\theta_{j+1} = \theta_j - \alpha \nabla_\theta J(\theta) \tag{8}$$

The parameter α is known as the **learning rate**. This difference equation describes **batch gradient decent**. When the training set is large, one common modification can be made

$$\theta_{j+1} = \theta_j - \alpha \nabla_\theta J^{(i++)}(\theta_j) \tag{9}$$

where the ++ operator indicates iteration through the training set with each global iteration. This is referred to as a **stochastic gradient descent**.

2.3 Probabilistic interpretation

In order to appreciate the choice of loss function in the ordinary-least-squares model, consider the "error" produced by a linear model in a training example

$$\begin{aligned} \epsilon^{(i)} &= y^{(i)} - h(x^i) \\ &= y^{(i)} - \theta^T x^{(i)} \end{aligned} \tag{10}$$

Let us assume that the distribution of $\epsilon^{(i)}$ in some training set is independently and identically distributed (IID) according to a Gaussian distribution

$$p(e^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{e^{(i)2}}{2\sigma^2}\right) \tag{11}$$

which by (10) implies

$$p(y^{(i)}|x^{(i)};\theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2\sigma^2}\right) \tag{12}$$

This assumption may alternatively be expressed

$$e^{(i)} \sim \mathcal{N}(0, \sigma^2) \tag{13}$$

Equation (12) states the conditional probability of the random variable $x^{(i)}$ given the random variable $y^{(i)}$, parameterised by θ, takes the form of the given

Gaussian distribution. Minimizing the error amounts to maximizing this probability. In order to prescribe this probability to a training set, we use matrix notation. The **likelihood** function of a training set can then be expressed

$$L(\theta) = L(\theta; X, \vec{y}) = p(\vec{y}|X;\theta) \tag{14}$$

where $\vec{y} = \left[y^{(1)}, y^{(2)}, \dots, y^{(N)}\right]^T$ Using the independence assumption

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{N} p(y^{(i)}|x^{(i)};\theta) \\ &= \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2\sigma^2}\right) \end{aligned} \tag{15}$$

We node that maximizing the likelihood function is equivalent to maximizing any monotonically increasing function of the likelihood. We choose the logarithm function[1] to yield

$$\begin{aligned} \ell(\theta) &\equiv \log L(\theta) \\ &= \sum_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2\sigma^2}\right) \\ &\sim \sum_{i=1}^{N} \frac{1}{2}\left(y^{(i)} - \theta^T x^{(i)}\right)^2 \end{aligned} \tag{16}$$

Not incidentally, we see from (6) that maximizing the likelihood (16) is equivalent to minimizing our loss function defined in (7).

2.3.1 Weighted linear regression

One common modification to the liner regression model is to account for *weights* in the loss function

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{N} w^{(i)}\left(y^{(i)} - h_\theta(x^{(i)})\right)^2 \tag{17}$$

A standard choice for $w^{(i)}$ is

$$w^{(i)} = \exp\left(-\frac{\left(x^{(i)} - x\right)^T \left(x^{(i)} - x\right)}{2\tau^2}\right) \tag{18}$$

where τ is the **bandwidth parameter** and

elaborate on x

[1] In computer arithmetic, addition is less expensive that multiplication. By using the logarithm function, products become sums and computation efficiency improves.

3 Logistic regression

Logistic regression is an approach to *binary classification* $\mathbb{Y} = \{0, 1\}$ using a classifier naturally based on the **logistic function**.

$$h(x;\theta) = \frac{1}{1 + e^{-\theta^T x}} \tag{19}$$

Probabilistically, we interpret h such that

$$h_\theta(x) = P(y = 1|x;\theta) \tag{20}$$

and hence the conditional probability of y given x takes the form

$$P(y|x;\theta) = h_\theta(x)^y + (1 - h_\theta(x))^{1-y} \tag{21}$$

Extending (21) for a training set, using the assumption that the target errors are independently and identically distributed

$$\begin{aligned} L(\vec{y}) &= p(X;\theta) \\ &= \prod_{i=1}^{N} \left(h_\theta(x^{(i)})\right)^{y^{(i)}} \left(1 - h_\theta(x^{(i)})\right)^{(1-y^{(i)})} \end{aligned} \tag{22}$$

calculation of log likelihood plus derivative

4 Generalised Linear Model

The models discussed in the previous section may be expressed as subsets of a generalised **exponential family** of distributions which posit the target error distribution according to an arbitrary statistical model.

$$p(y;\eta) = b(y)\exp\left(\eta^T T(y)\right) - \alpha(\eta) \tag{23}$$

Define terms and express following distributions as GLM

4.1 Gaussian distribution

4.2 Bernoulli distribution

4.3 Multinomial distribution

4.4 Poisson distribution

4.5 Gamma distribution

4.6 Dirichlet distribution

5 Constructing a Generalised Linear Model

Lay out strategy for designing. Exemplify with OLS, logistic regression, softmax regression.

5.1 Ordinary least squares

5.2 Logistic regression

5.3 Softmax regression

6 Generative learning

A *generative model* aims to model data with a predictive capacity for both the feature and target variables of a data set. This is in contrast to *discriminative* models such as the regression models in the previous section which aim to model the target variable based on input features. Probabilistically, we may interpret generative learning as determining a probability distribution of $x \in \mathbb{X}$ given $y \in \mathbb{Y}$

$$p(x|y) \tag{24}$$

6.1 Gaussian discriminative analysis

6.2 Naive Bayes

The Naive Bayes assumption states that the conditional probability of individual elements in a feature set given the target are *independent.* That is to say, for $x \in \mathbb{R}^{\{}n+1\}$,

$$p(x|y) \equiv p(x_1, x_2, \ldots x_{n+1}|y) = \prod_{i=1}^{n+1} p(x_i|y) \tag{25}$$

Alternatively, this may be more intuitively expressed

$$p(x_i|y) = p(x_i|y, x_j) \qquad \forall i, j \in \{1, 2, \ldots, n+1\} \tag{26}$$

6.3 Learning Theory

6.3.1 Training error

The *training error* of a classifier describes its failure ratio in predicting output $y^{(i)}$ in the training set

$$\xi_n(h) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}\left[h(x^{(i)}) \neq y^{(i)}\right] \tag{27}$$

where the operator $\mathbb{I}$ maps a proposition to the binary value corresponding to its truth value.

$$\mathbb{I}\left[true\right] = 1 \qquad \mathbb{I}\left[false\right] = 0$$

A small training error does not necessarily an indication of the quality of a classifier. A classifier may have a low training error but may not generalise well to data outside the training set. Such a classifier is the result of *overtraining.*

6.3.2 Generalization error

Generalization is the application of a classifier to data beyond the training set. The *generalisation error* measures the failure ratio of predicting $y^{(i)}$ outside the training set. A good quality classifier is one that generalises well.

$$\xi(h) = \frac{1}{n'} \sum_{i=n}^{n+n'} \left[h(x^{(i)}) \neq y^{(i)} \right] \tag{28}$$

7 Linear classification

A linear classifier demarcates two classes within a feature space by means of a hyperplane known as a *decision boundary.* Objects are classified according to which side of the hyperplane they lie.

Definition 7.1. For a feature space $\mathbb{X} \in \mathbb{R}^{d+1}$, a decision boundary is defined by the normal vector $\theta \in \mathbb{R}^{d+1}$

$$P_\theta = \{z | z \cdot \theta \ = 0\} \tag{29}$$

The side to which a point $x \in \mathbb{R}^d$ lies may be qualified by

$$\text{sign}(x \cdot \theta) \equiv x \cdot \theta / ||x|| ||\theta|| \tag{30}$$

Thus, we may express an arbitrary binary linear classifier $h : \mathbb{R}^{d+1} \to \{+1, -1\}$ as

$$h(x, \theta) = \text{sign}(x \cdot \theta) \tag{31}$$

7.1 Separability

A training set $T = \{(x_i, y_i) | i = 1, 2 \dots n\}$ is said to be *linearly separable* if there exists a decision boundary that correctly classifies every example (x_i, y_i)

$$\exists \theta \forall (y, x)[y(x \cdot \theta) > 0] \tag{32}$$

7.2 Mistake-driven algorithms

Mistake driven algorithms are simple procedures used *learn* the parameter θ of a linear model for some training set.

7.2.1 Perceptron

The perceptron algorithm iterates through the training set, modifying θ accordingly each time it miss-classifies a training example.

```
def perceptron(training_set, epochs, init_theta):
    theta = init_theta
    for n in range(epochs):
        for x, y in training_set:
            if y * theta.dot(x) <= 0:
                theta += x
    return theta
```

If training set is linearly separable, perceptron is guaranteed to converge to solution. Prove this!

7.3 Loss functions

In the perceptron algorithm, every misclassification triggers a modification of equal significance to the parameter θ. This disregards the distance of the training example from the decision boundary. A somewhat more sophisticated approach would be to incorporate some *loss function* that accounts for the margin of error in a misclassification an modifies θ accordingly.

7.3.1 Hinge Loss

Hinge loss gives penalizes θ on each iteration proportionally to the margin of error

$$\text{Hinge}(x, y; \theta) = \max\{0, 1 - y(x \cdot \theta)\} \tag{33}$$

```
def hingeloss(training_set, epochs, init_theta):
    theta = init_theta
    for n in range(epochs):
        for x, y in training_set:
            theta += max(0, 1 - y * theta.dot(x))
    return theta
```

7.3.2 Passive-aggressive Algorithm

Large margin classifiers / Passive aggressive algorithm

7.4 Support Vector Machines

8 Recommender problems

9 Non-linear classification

One approach to non-linear classification involves performing a linear classification on the set of vectors resulting from some non-linear map of the feature set

$$\varphi : \mathbb{R}^d \to \mathbb{R}^k \tag{34}$$

where typically $k > d$. In effect, the training set becomes

$$T \to T'_n = \{(\varphi(x_i), y_i) | i = 1, 2 \ldots n\} \tag{35}$$

The resulting linear decision boundary is mapped into a non-linear boundary after returning to the original coordinates.

9.1 Kernel methods

Feature maps in (34) typically increase the dimension of the feature space and thus bear an inflated computational cost. A method used to address this involves circumventing computations with the high dimensional vectors in favour of **inner products** between them.

$$\varphi(x) \cdot \varphi(x') = K(x, x') \tag{36}$$

K is known as a *kernel function*. A kernel function is *valid* if there exists some φ such that (36) holds.

9.1.1 Kernel perceptron

The kernel perceptron can be understood by consider θ expressed in the form

$$\theta = \sum_{i=1}^{n} \alpha_i y_i \varphi(x_i) \tag{37}$$

where α_i denotes the number of times training example i was missclassified. Taking the inner product with $\varphi(x_j)$ on both sides

$$\theta \cdot \varphi(x_j) = \sum_{i=1}^{n} \alpha_i K(x_i, x_j) \tag{38}$$

kernel perceptron pseudocode

9.1.2 Kernel Functions

4 theorems for kernel composition

10 Neural Networks

In the context of the methods discussed thus far, learning amounts to obtaining an hypothesis that maps elements in a feature space to predictions in the solution space

$$h : \mathbb{R}^d \to \mathbb{Y} \tag{39}$$

where for regression $\mathbb{Y} = \mathbb{R}$ and for classification $\mathbb{Y}$ is a small and finite. Neural networks are a class of hypotheses that are the functional composition of multiple *layers*

$$h = \ell_1 \circ \ell_2 \circ \cdots \circ \ell_n \tag{40}$$

where n is the number of layers. Each layer is a map

$$\ell_i : \mathbb{R}^{k_{i-1}} \to \mathbb{R}^{k_i} \tag{41}$$

where $k_i \in \mathbb{N}$, $k_0 = d$ and $\mathbb{R}^{k_n} \equiv \mathbb{Y}$. A layer itself is composed of an *activation* together with a transformation function

$$\ell_i = \alpha_i \circ \beta_i \tag{42}$$

The transformation function maps the vectors through spaces which, in general, vary in dimension between layers.

$$\beta_i : \mathbb{R}^{k_{i-1}} \to \mathbb{R}^{k_i} \tag{43}$$

Using the notation $\mathbb{M}(m, n)$ to denote an $m \times n$ real matrix, the transformation takes the form

$$\beta_i(x; W_i, b_i) = W_i x + b_i \tag{44}$$

where the weights W_i, b_i are parameters

$$x \in \mathbb{R}^{k_{i-1}} \quad W_i \in \mathbb{M}(k_{i-1}, k_i) \quad b_i \in \mathbb{R}^{k_i} \tag{45}$$

The activation is inspired by and typically mimics the role of *threshold potential* in a biological neuron

$$\alpha_i : \mathbb{R}^{k_i} \to \mathbb{R}^{k_i} \tag{46}$$

Examples of commonly used activation functions include the sigmoid function (47), ReLU (48), hyperbolic tangent (49).

$$\alpha(x) = \frac{1}{1 + \exp(x)} \tag{47}$$

$$\alpha(x) = \begin{cases} 0 & \text{for} \quad x \leq 0 \\ x & \text{for} \quad x > 0 \end{cases} \tag{48}$$

$$\alpha(x) = \tanh(x) \tag{49}$$

A neural network is parameterized by the set of weights

$$\Theta = \{W_i, b_i | i = 1, 2, \ldots n\} \tag{50}$$

The set of hypotheses for a particular neural network can may compactly be expressed

$$\mathcal{H} = h(x; \Theta) \tag{51}$$

Finding a $h \in \mathcal{H}$ that generalises well from a given training set amounts learning the parameter Θ. This is the objective of *training*.

10.1 Training neural networks