Machine Learning
Lecture 8

Lecture by Dr. Shai Fine
Typeset by Steven Karas

2017-12-10
Last edited 21:04:08 2017-12-10

Disclaimer These lecture notes are based on the lecture for the course Machine Learning, taught by Dr. Shai Fine at IDC Herzliyah in the fall semester of 2017/2018. Sections may be based on the lecture slides written by Dr. Shai Fine.

Agenda

- Continuous metric feature space
- Classes of Linear Models
- Decision boundaries:
  - Discriminant functions - Linear, Generalized Linear
  - Rosenblatt's Perceptron
  - Widrow-Hoff Least Mean Square
  - MSE and Closed form

1 Linear Models

We always assume that we're working in a continuous feature space.

1.1 Linear Decision Boundaries

Methods that give a linear decision boundary: $\{x \mid w^\intercal x + w_0 = 0\}$

Generally two approaches: either model the boundary as linear, or a discriminant function for each class.

1.2 Discriminant function

$$f(x_1, \ldots, x_d) : \mathbb{R}^d \to \mathbb{R}$$

This function defines a hypersurface that partitions the space into regions. The nuclei $f(x) = 0$ is called the decision surface.

Classification is simply checking if the value of this function is positive or negative.

Linear discriminant functions

$$g(x \mid w, w_0) = w^\intercal x + w_0 = \sum_{j=1}^{d} w_j x_j + w_0$$

Where w is the weight vector, and w_0 is the bias weight. This defines a hyperplane $g(x \mid w, w_0) = 0$ in $\mathbb{R}^d$. Note that w is normal to any vector lying in the hyperplane.

If we define the $d+1$ dimensional vectors $\tilde{x} = (x, 1)$ and $\tilde{w} = (w, w_0)$, we can rewrite the discriminant function as $g(\tilde{x} \mid \tilde{w}) = \tilde{w}^\intercal \tilde{x}$. Note that in this notation, $g(\tilde{x} \mid \tilde{w}) = 0$ defines a hyperplane that passes through the origin in $\mathbb{R}^{d+1}$.

The advantage of this method is that we get an $O(d)$ space/time computation.

2 Perceptron

W.S. McCulloch and W. Pitts in 1943 presented "A logical calculus of the ideas immanent in nervous activity" which described neurons.

$$x_i \cdot w_{x_i} + y_i \cdot w_{y_i} - 1 \cdot w_{b_i} \Rightarrow \begin{cases} 0 & \text{if sum} < 0 \\ 1 & \text{else} \end{cases}$$

Frank Rosenblatt presented in 1962 "Principles of Neurodynamics", which described the perceptron model, which used neurons to classify general vectors:

$$\sum_{i=0}^{k} input_i \cdot weight_i \Rightarrow \begin{cases} 1 & \text{if} > \text{threshold} \\ -1 & \text{if} < \text{threshold} \end{cases}$$

Marvin Minsky and Seymour Papert published in 1969 "Perceptrons", which described the limitations of the perceptron model.

Algorithm The Perceptron rule is controlled by a learning rate η.

Perceptron Rule

```
w⃗ := w⃗_0
ErrorFound := True
while ErrorFound:
  ErrorFound := False
  for i in 1..n:
    ŷ_i := sign(x⃗_i · w⃗_t)
    if ŷ_i ≠ y_i:
      ErrorFound := True
      w⃗ := w⃗ + ηy_i x⃗_i

return w⃗
```

This algorithm assumes the $\mathbb{R}^{d+1}$ vector notation

Perceptron Criterion

$$E^{\text{perc}}(w) = -\sum_{i \in D_{\text{miss}}} w^{\intercal}(x_i t_i)$$

Where D_{miss} is the set of samples that were misclassified by w, and t_i is the true classification of the x_i samples.

The gradient of this error function can be used to minimize over multiple training rounds:

$$\nabla_w E^{\text{perc}} = \left[\frac{\partial E^{\text{perc}}(w)}{\partial w_1}, \ldots, \frac{\partial E^{\text{perc}}(w)}{\partial w_d}\right]; \quad \frac{\partial E^{\text{perc}}(w)}{\partial w_j} = -\sum_{i \in D_{\text{miss}}} x_{i_j} t_i$$

Stochastic Gradient Descent

$$w_{\text{new}} = w + \eta \Delta w = w - \eta \nabla_w E^{\text{perc}}$$

$$w_{\text{new}_j} = w_j + \eta \Delta w_j = w_j + \eta(t_i - o_i) x_{i_j}$$

Where o_i is the output of the ith instance.

If the training data is linearly separable, this converges in a finite number of steps.

Transfer function Effectively, the perceptron is a neural network with a linear transfer function. Nonlinear transfer functions allow such networks to learn nonlinear models, but this is out of scope for this lecture.

3 Widrow-Hoff Algorithm

Also called the Least Mean Square (LMS) algorithm. Also called the Adaptive Linear Neuron (ADALINE) network and its learning rule.

This minimizes the mean square error (MSE): [1]

$$E(w) = \frac{1}{2}\sum_{i\in D}(t_i - w^\intercal x_i)^2 = \frac{1}{2}\left[\underbrace{\sum_{i\in D^+}(w\cdot x_i - 1)^2}_{\text{minimize dist to +1 isoline}} + \underbrace{\sum_{i\in D^-}(w\cdot x_i + 1)^2}_{\text{minimize dist to -1 isoline}}\right]$$

Where D is the training set. We can train on the gradient of the error:

$$\frac{\partial E}{\partial w_j} = \frac{1}{2}\sum_{i\in D} 2(t_i - w^\intercal x_i)\frac{\partial}{\partial w_j}(t^i - w^\intercal x_i) = \sum_{i\in D}(t_i - w^\intercal x_i)(-x_{i_j})$$

Which gives us a learning rule:

$$w_{\text{new}_j} = w_j + \eta\Delta w_j = w_j + \eta(t_i - w^\intercal x_i)x_{i_j}$$

This can still only learn linear models.

3.1 Closed form

We can train the model in closed form by solving:

$$X\cdot\vec{w} = \vec{t}$$

If X is singular, we can write $\vec{w} = X^{-1}\vec{t}$. Most of the time, $X_{n\times d}$ is singular (e.g. $n > d$ - more samples than features)

The error vector $e = X\vec{w} - \vec{t}$ can be minimized:

$$E(\vec{w}) = ||X\vec{w} - \vec{t}||^2 = \sum_{i=1}^{n}(w^\intercal x_i - t_i)^2$$

$$\nabla E(\vec{w}) = \sum_{i=1}^{n} 2(w^\intercal x_i - t_i)x_i = 2X^\intercal(X\vec{w} - \vec{t})$$

Pseudoinverses The $d\times d$ matrix $(X^\intercal X)^{-1}X^\intercal$ is called the pseudoinverse[2] of X.

This allows us to minimize the derivative:

$$X^\intercal X\vec{w} = X^\intercal\vec{t} \Rightarrow \vec{w} = (X^\intercal X)^{-1}X^\intercal\vec{t}$$

4 Dimensionality Reduction

Motive Fewer dimensions reduces the time complexity (some algorithms are polynomial w.r.t. the number of features, some are even worse). Curse of dimensionality means every is sparse, and simpler models are more robust on smaller dimensions. In practice, we find that high dimensionality actually hurts.

4.1 Linear Projection

Projections are linear transformations on the data from one axis system to another.

Translation Subtracting the mean vector from all vector points moves the origin to the center of the data:

$$X - \mu$$

Rotation Project on the axis using a dot product: $(x_i - \mu)\cdot a_j$.

[1] This was accompanied by a rather bad example in the slides

[2] also called the Moore-Penrose inverse

Transformation Matrix

$$A = [a_1, \ldots, a_k] \in \mathbb{R}^{d \times k}$$

We define these to be orthonormal: $A^\intercal A = I$.

4.2 Lagrange Multiplier

Find the extremum of a function $f(x_1, x_2)$ subject to a constraint $g(x_1, x_2) = 0$.

$$L(x_1, x_2, \lambda) = f(x_1, x_2) - \lambda g(x_1, x_2)$$

This function behaves like the original function if the constraint is fulfilled. Note the gradient:

$$\nabla_{x_1,x_2,\lambda} L(x_1, x_2, \lambda) = 0 \Leftrightarrow \begin{cases} \nabla_{x_1,x_2}[f(x_1, x_2) - \lambda g(x_1, x_2)] = 0 \\ g(x_1, x_2) = 0 \end{cases}$$

This allows us to maximize $f(x_1, x_2)$ subject to the constraint by finding the largest value c such that the level curve (contour) $f(x_1, x_2) = c$ intersects $g(x_1, x_2) = 0$. This happens when the curves have a common tangent.

Because the gradient is perpendicular to the contour, the contours of f and g are parallel iff the gradients of f, g are parallel.

We want the points where $g(x_1, x_2) = 0$ and $\nabla_{x_1,x_2} f(x_1, x_2) = \lambda \nabla_{x_1,x_2} g(x_1, x_2)$. The langrangian is used to control the magnitude of the gradient vectors. [3]

4.3 Principal Component Analysis (PCA)

Projects the data along the axes where the data lies. The idea is to project the data into a lower dimension that maximizes the information (by minimizing the reconstruction error)

Reconstruction Error

$$\min_{G \in \mathbb{R}^{d \times k}} \underbrace{||X - G(G^\intercal X)||^2}_{||X - \hat{X}||^2} \text{ subject to } G^\intercal G = I_k$$

Theory Let $z_1 = a_1^\intercal X$ be the data projected onto the first principal component:

$$\text{Var}[z_1] = E[z_1^2] = E[(a_1^\intercal X)^2] = a_1^\intercal E[X X^\intercal] a_1 = a_1^\intercal \Sigma a_1$$

Where Σ is the covariance matrix of X and the data is mean adjusted.

Find a_1 that maximizes $\text{Var}[z_1]$ subject to the constraint $a_1^\intercal a_1 = 1$. Solving this constraint optimization problem using Lagrange Multiplier.

$$\max L(a_1, \lambda) = a_1^\intercal \Sigma a_1 - \lambda(a_1^\intercal a_1 - 1)$$

Take the derivative with respect to a_1

$$2(\Sigma a_1 - \lambda a_1) = 0 \Rightarrow \Sigma a_1 = \lambda a_1$$

This means that a_1 is the eigenvector of Σ and λ is the eigenvalue by definition.

$$a_1^\intercal \Sigma a_1 = a_1^\intercal (\lambda a_1) = \lambda a_1^\intercal a_1 = \lambda$$

Which means we're actually trying to maximize the eigenvalue.

The rest of the principle components are simply the eigenvectors sorted by descending eigenvalues. [4]

Algorithm Given a data set $X \in \mathbb{R}^{d \times N}$. Mean adjust and normalize the data. Approximate the covariance matrix with the scatter matrix: $S = \frac{1}{n} \Sigma_i x_i x_i^\intercal$

Compute the eigenvectors and eigenvalues. Sort the eigenvectors in descending order by eigenvalue. Select the k first eigenvectors.

The projected data is $Z \in \mathbb{R}^{k \times N} = A^\intercal X$ where $A = [a_1, \ldots, a_k]$.

[3] There is a fully worked example on slide 33

[4] Full proof on slide 12

Selection of k

$$\text{Proportion of variance (PoV)} = \frac{\sum_{j=1}^{k} \lambda_j}{\sum_{j=1}^{d} \lambda_j} > 0.9$$

Scree graphs plot the PoV vs k. k is typically selected by stopping at an elbow, where most of the information is preserved. This is typically at a much lower dimension.

5 Next week

There will not be a lecture next week due to Hannukah. Happy Hannukah!

We will discuss bayesian classifiers next lecture.