Fundamentals of Algorithm Design and Machine Learning

Sandeep Tyagi: 24BM6JP47
Instructor: Dr. Aritra Hazra
Department of Computer Science Engineering,
Indian Institute of Technology, Kharagpur

Week 9 summary scribe (17-21 Mar-25)

1 Previously on Linear Classification

The primary goal of linear classification problems is to create a hyperplane that differentiates two classes. A linear classifier aims to find a single line that separates these classes effectively.

For a given input $\mathbf{x}$ with d features, classification is performed as follows:

$$y = \{\, 1\ , if sign(\mathbf{W}^T\mathbf{x}) > 0 - 1, if sign(\mathbf{W}^T\mathbf{x}) < 0$$

where $\mathbf{W}$ represents the weights defining the hyperplane. A similar method applies to linear regression, where the predicted value is given by:

$$y = \mathbf{W}^T\mathbf{x}$$

The in-sample error, or the error during training, is computed as:

$$E_{in} = \frac{1}{N}\sum(y - \mathbf{W}^T\mathbf{x})^2$$

To minimize this error, we compute its gradient with respect to $\mathbf{W}$ and set it to zero, leading to the optimal weight computation:

$$\mathbf{W} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

An interesting observation here is that the optimal weights can be computed in a single step, eliminating the need for iterative updates.

2 Variants of Linear Classification

In our previous discussion, we explored how data points in a dataset can be linearly separable, allowing us to compute an optimal hyperplane for classification. However, in many cases, data points may not be linearly separable.

To address this, we transform the original feature set into a new feature space, denoted as:

$$\mathbf{z} = \Phi(\mathbf{x})$$

where $\Phi(\mathbf{x})$ represents a transformation that makes the new features linearly separable. For example, a dataset with features $\mathbf{x}$ can be transformed into a new feature set where:

$$z = x_1^2$$

In this transformation process, multiple options exist. A general approach is to include all quadratic terms of the original features, leading to an expanded feature set:

$$(1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$$

However, this expanded feature set has certain disadvantages:

1. It is highly generic, leading to potential overfitting on the training set.
2. Computing and storing all features is computationally expensive, increasing the model's complexity.

To mitigate these issues, we can reduce the number of features, for instance:

$$(1, x_1, x_2) = (1, x_1^2 + x_2^2)$$

which transforms the data distribution. Further reduction can yield:

$$(1, x_1x_2) = (x_1^2 + x_2^2)$$

with a classification rule such as:

$$if x_1^2 + x_2^2 < 0.6, then -1, else +1$$

Such transformations may yield better classification results than using raw features. However, an important aspect to note is that this transformation process must be manually designed by the user by analyzing data patterns. This practice, known as **Data Snooping**, has a significant drawback—rather than the machine learning the data, the user manually discovers patterns, which can lead to biased feature selection.

3 Variants of Linear Regression

Linear regression aims to fit a single line to all data points in a dataset. However, a useful variation is to divide the training set into small neighborhoods and fit a separate linear regression model for each local interval. This approach is known as **Locally Weighted Regression (LWR)**.

The objective remains the same as standard linear regression—minimizing the squared error between predictions and actual values. However, instead of considering all data points equally, we assign higher weights to points closer to the target x using a kernel function. This modifies the cost function to:

$$\min_{\mathbf{W}} \sum_{i=1}^{n} w_i (\mathbf{W}^T \mathbf{x}_i - y_i)^2$$

where w_i is a weight assigned to each data point, determined by a Gaussian kernel:

$$w_i = \exp\left(-\frac{(x_i - x)^2}{2\tau^2}\right)$$

Here, τ is the bandwidth parameter that controls the influence of nearby points. A smaller τ means only very close points contribute significantly, while a larger τ allows for a broader influence.

If we visualize w_i as a function of x, it forms a bell-shaped curve, indicating that the influence of a data point x_i decreases as we move farther from the point of interest x.

This method provides greater flexibility in capturing local patterns in the data compared to standard linear regression.

4 Logistic Regression

In any classification problem, the primary objective is to obtain a function that maps values between 0 and 1. However, we need a specific function $f(s)$ that exhibits a smooth transition, ensuring differentiability and a gradient for optimizing predictions.

One such function that satisfies these properties is the **sigmoid function**:

$$\sigma(s) = \frac{e^s}{1 + e^s}$$

A key property of the sigmoid function is:

$$\sigma(-s) = 1 - \sigma(s)$$

Our goal is to fit the input into this function so that the output can be interpreted as a probability. Thus, our hypothesis function is defined as:

$$H(s) = \sigma(s)$$

where

$$s = \mathbf{W}^T \mathbf{x}$$

To estimate the model parameters, we use the **Maximum Likelihood Estimation (MLE)**. The probability of a given class label y given the input $\mathbf{x}$ is:

$$P(y \mid \mathbf{x}) = \{ \, h \, (\mathbf{x}), if y = +11 - h(\mathbf{x}), if y = -1$$

which can be rewritten as:

$$P(y \mid \mathbf{x}) = \sigma(s)^y \cdot \sigma(-s)^{1-y}$$

To obtain the MLE, we maximize the likelihood function, which is the product of all instance probabilities:

$$\max \prod_{i=1}^{N} \sigma(y_i \mathbf{W}^T \mathbf{x}_i)$$

Taking the negative log-likelihood, we obtain the in-sample error function:

$$E_{in} = \frac{1}{N} \sum_{i=1}^{N} \ln(1 + e^{-y_i \mathbf{W}^T \mathbf{x}_i})$$

The objective now is to minimize E_{in} iteratively. Since the error function is smooth, we can update the weights using **gradient descent**:

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \eta \nabla E_{in}(\mathbf{W})$$

where η is the learning rate. A key observation here is that the weight updates are adaptive—initially, larger steps are taken, but as we approach the minimum, the step size naturally decreases.

This adaptive weight adjustment ensures convergence to the optimal parameters for logistic regression.

5 Linear Classification: Perceptron (Linear Discriminator)

In a linear classification problem, we aim to find an unknown function f that maps an attribute set $\mathbf{X}$ to an output Y, such that:

$$f : \mathbf{X} \to Y, \quad where Y \in \{+1, -1\}$$

If the data is well-separated by a single linear boundary, a **Perceptron** serves as an appropriate model. However, if the data is not linearly separable, either a different machine learning algorithm is required or multiple linear

classifiers can be stacked in a staggered fashion to form an **Artificial Neural Network (ANN)**.

Perceptron Learning Rule

The perceptron is based on a simple thresholded sum of weighted inputs. The decision function is:

$$h(\mathbf{x}) = sign\left(\sum_{i=1}^{d} w_i x_i\right)$$

To update the weights, we use the weight update rule:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \Delta\mathbf{w}$$

where the weight adjustment $\Delta\mathbf{w}$ is determined by minimizing the in-sample error E_{in}. Differentiating E_{in} with respect to w_k:

$$\frac{\partial E_{in}}{\partial w_k} = \sum_{n=1}^{N}\left(y_n - sign\left(\sum_{i=1}^{d} w_i x_{ni}\right)\right) \cdot sign(x_{nk})$$

Defining y_n as the target output t_n and the predicted output as $o_n = sign\left(\sum_{i=1}^{d} w_i x_{ni}\right)$, the equation simplifies to:

$$\frac{\partial E_{in}}{\partial w_k} = \sum_{n=1}^{N}(t_n - o_n)x_{nk}$$

Thus, the weight update rule becomes:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \eta\sum_{n=1}^{N}(t_n - o_n)x_{nk}$$

where η is the learning rate. Since the weight update is computed over all training instances before updating, this follows a **Batch Gradient Descent** approach.

Optimization Objective

To optimize the perceptron, we minimize the in-sample error:

$$E_{in}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left[y_n - sign\left(\sum_{i=1}^{d} w_i x_{ni}\right)\right]^2$$

Since this function is piecewise constant, we iteratively adjust feature weights based on E_{in} to improve classification performance.

6 Artificial Neural Node

An **Artificial Neural Node** consists of a summing unit followed by a soft threshold function to compute the hypothesis $h(\mathbf{x})$. This function, denoted as $\sigma(s)$, can be defined as:

$$\sigma(s) = \tanh(s) \quad or \quad \sigma(s) = \frac{1}{1+e^{-s}}$$

or any other smooth activation function.

6.1 Learning Boolean AND

The Boolean AND function can be realized using a **single-layer Perceptron**. Given input features x_1 and x_2 with weights w_0, w_1, and w_2, the perceptron hypothesis is:

$$h(x) = \sigma(w_0 + w_1 x_1 + w_2 x_2)$$

To implement an AND gate, we need to learn appropriate weights that satisfy the following truth table:

x_1	x_2	AND (y)
0	0	0
0	1	0
1	0	0
1	1	1

By setting weights such as $w_0 = -1.5$, $w_1 = 1$, and $w_2 = 1$, we get:

$$h(x) = \sigma(-1.5 + x_1 + x_2)$$

which correctly classifies the AND function.

6.2 Learning Boolean OR

Similarly, the OR function can also be implemented using a **single-layer Perceptron**. By slightly adjusting the bias weight w_0, we obtain the following truth table:

x_1	x_2	OR (y)
0	0	0
0	1	1
1	0	1
1	1	1

By setting $w_0 = -0.5$, $w_1 = 1$, and $w_2 = 1$, the perceptron equation becomes:

$$h(x) = \sigma(-0.5 + x_1 + x_2)$$

which correctly models the OR function.

6.3 Learning Boolean XOR

Unlike AND and OR, the XOR function **cannot be solved by a single-layer Perceptron** because it is not linearly separable. The truth table for XOR is:

x_1	x_2	XOR (y)
0	0	0
0	1	1
1	0	1
1	1	0

To implement XOR, we need a **multi-layer perceptron (MLP)** with a hidden layer. A possible approach is to use an intermediate hidden layer computing AND and OR separately:

$$h_1 = \sigma(w_0^{(1)} + w_1^{(1)} x_1 + w_2^{(1)} x_2) \quad (ANDcomponent)$$

$$h_2 = \sigma(w_0^{(2)} + w_1^{(2)} x_1 + w_2^{(2)} x_2) \quad (ORcomponent)$$

$$y = \sigma(w_0^{(3)} + w_1^{(3)} h_1 + w_2^{(3)} h_2)$$

With appropriate weights, this structure can successfully compute XOR.
article amsmath, amssymb, graphicx
Backpropagation Algorithm

7 Backpropagation Algorithm

7.1 Introduction

Backpropagation is an optimization technique used for training artificial neural networks by adjusting the weights iteratively to minimize the error function. The algorithm consists of forward propagation, error computation, and backward propagation to update weights.

7.2 Algorithm Steps

1. Initialize all weights $w_{i,j}^{(l)}$ randomly.
2. For each iteration $t = 0, 1, 2, \ldots$:

(a) Pick a training example $n \in \{1, 2, \ldots, N\}$.

(b) Perform forward propagation to compute all activations $x_j^{(l)}$.

(c) Compute the error for the output layer.

(d) Perform backward propagation to compute all error terms $\delta_j^{(l)}$.

(e) Update the weights using:

$$w_{i,j}^{(l)} \leftarrow w_{i,j}^{(l)} - \eta \cdot x_i^{(l-1)} \cdot \delta_j^{(l)} \tag{1}$$

where η is the learning rate.

3. Repeat the process until convergence.

7.3 Mathematical Formulation

The weight update rule in backpropagation is derived from the chain rule of differentiation. Given an error function $E(w)$, the weight update is computed as:

$$\frac{\partial E}{\partial w_{i,j}^{(l)}} = \frac{\partial E}{\partial s_j^{(l)}} \cdot \frac{\partial s_j^{(l)}}{\partial w_{i,j}^{(l)}} \tag{2}$$

where:

$$s_j^{(l)} = \sum_i w_{i,j}^{(l)} x_i^{(l-1)} \tag{3}$$

and the gradient of the activation function $\sigma(s)$ is given by:

$$\delta_j^{(l)} = \sigma'(s_j^{(l)}) \sum_k w_{j,k}^{(l+1)} \delta_k^{(l+1)} \tag{4}$$

7.4 Stopping Criteria

The training stops when one of the following conditions is met:

- The error $E(w)$ falls below a predefined threshold.
- A maximum number of iterations is reached.
- The change in error between iterations is sufficiently small.

8 Linear Discriminant Analysis (LDA)

- Among all possible linear discriminators, the goal is to find the best one for classification.
- Any linear discriminator is represented as:

$$w_1 x_1 + w_2 x_2 + b = 0.$$

- For any new point $x_n = \langle a_1, a_2 \rangle$:

$$w_1 a_1 + w_2 a_2 + b \geq 0, \quad (Class + 1)$$
$$w_1 a_1 + w_2 a_2 + b < 0, \quad (Class - 1)$$

- The best discriminator is the one that passes through the middle of both classes.

8.1 Defining the Middle Line

- For a training point x_i, the perpendicular distance from the discriminator line is given by:

$$d_i = \frac{w_1 x_{i1} + w_2 x_{i2} + b}{\sqrt{w_1^2 + w_2^2}}.$$

- The goal is to maximize the minimum distance of the points from the discriminator:

$$\max \left[\min_i \frac{w_1 x_{i1} + w_2 x_{i2} + b}{\sqrt{w_1^2 + w_2^2}} \right].$$

- The optimization depends only on the numerator.
- The values of w_1, w_2, b are chosen such that for the closest point, the minimum distance (numerator) is set to 1.
- Hence, the optimization problem simplifies to:

$$\max \left(\frac{1}{\sqrt{\|W\|}} \right),$$

where $\|W\| = W^T W$.

- This ensures that the minimum distance of any point from the discriminator is 1:

$$y_i(w_1 x_{i1} + w_2 x_{i2} + b) \geq 1.$$

9 Primal Optimization Problem

- The optimization problem is formulated as:

$$\min \left(\frac{1}{\sqrt{\|W\|}} \right) = \max \left(\frac{0.5}{\sqrt{W^T W}} \right),$$

subject to:

$$y_i(W^T X_i + b) \geq 1, \quad \forall i.$$

- At any point in time, there exist **two support points** that define the optimal decision boundary.
- This leads to the concept of the **Support Vector Machine (SVM)**.

10 Dual Optimization Problem

- To solve the primal problem efficiently, we convert it into its dual form using Lagrange multipliers α_i.
- The optimization problem becomes:

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j X_i^T X_j,$$

  subject to:

$$\sum_i \alpha_i y_i = 0, \quad \alpha_i \geq 0 \quad \forall i.$$

- The optimal weight vector is given by:

$$W = \sum_i \alpha_i y_i X_i.$$

- Only support vectors (where $\alpha_i > 0$) contribute to the decision boundary.

10.1 What does Lagrange's Multiplier Try to Do?

- The Lagrange multipliers α_i scale the points X_i and their labels y_i, effectively finding the resultant vector from them.
- Not all $\alpha_i > 0$; only support vectors have nonzero α_i, making the method computationally efficient.
- Once W is found, b is easily computed:

$$W^T X + b = 1 \quad \Rightarrow \quad b = 1 - W^T X.$$

The **maximized Lagrangian function** is:

$$L = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (X_i^T X_j).$$

Since $\sum_{i=1}^{N} \alpha_i y_i = 0$, we rewrite L as:

$$L = U^T \alpha - \frac{1}{2} \alpha^T H \alpha.$$

Here, H is the **Hessian matrix**, where:

$$H_{ij} = y_i y_j (X_i^T X_j).$$

The problem is solved using **Quadratic Programming**, yielding:

$$W = \sum_{i=1}^{N} \alpha_i y_i X_i, \quad b = 1 - W^T X.$$