SMAI-S25-L17: Perceptron, Logistic Regression and SVM

C. V. Jawahar

IIIT Hyderabad

March 18, 2025

Recap:

- Problems of interest:
    - Learn a function $y = f(\mathbf{W}, \mathbf{x})$ from the data.
        - (a) Classification (b) Regression
    - Learn Feature Transformations $\mathbf{x}' = \mathbf{W}\mathbf{x}$ or $\mathbf{x}' = f(\mathbf{W}, \mathbf{x})$
        - (a) Feature Normalization (b) PCA
- Algorithms/Approaches:
    - Nearest Neighbour Algorithm
    - Linear Classification: $sign(\mathbf{w}^T\mathbf{x})$
    - Decide as ω_1 if $P(\omega_1|\mathbf{x}) \geq P(\omega_2|\mathbf{x})$ else ω_2.
    - Linear Regression: (a) closed form and (b) GD
    - Perceptron
- Supervised Learning:
    - Notion of Training, Validation and Testing; Performance Metrics
    - Data and Tools: Low rank data matrix, Distribution of Data; SVD
    - Notion of Loss Function, (eg. MSE), Regularization.
    - Role of Optimization, Convex and non-Convex optimization
    - Closed form solution, Gradient Descent, Eigen vector solns.
    - MDL, Model complexity, Overfitting

Recap of Perceptron

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

Where:

- $t = c(\vec{x})$ is target value
- o is perceptron output
- η is small constant (e.g., 0.1) called <u>learning rate</u>

Perceptron training rule

Can prove it will converge

- If training data is linearly separable
- and η sufficiently small

Understanding perceptrons as GD?

$$J' = \sum_{i=1}^{N} (t_i - o_i)^2$$

with $o_i = sign(\mathbf{w}^T \mathbf{x}_i)$? There is a problem. With small changes in the **w** or the line, the J is not changing. Let us now define:

$$J = \sum_{i=1}^{N} (t_i - o_i)^2 (-\mathbf{w}^T \mathbf{x}_i)$$

This additional terms pulls (or pushes) *proportionally*.
Let us now rewrite this J as sum of two parts one over $\mathcal{E}$ and the other on not in $\mathcal{E}$.

$$J = J_1 + J_2 = \sum_{\mathbf{x_i} \in \mathcal{E}} (t_i - o_i)^2 (-\mathbf{w}^T \mathbf{x}_i) + \sum_{\mathbf{x}_i \, not \in \mathcal{E}} (t_i - o_i)^2 (-\mathbf{w}^T \mathbf{x}_i)$$

Understanding perceptrons as GD?

$$J = \sum_{i=1}^{N} (t_i - o_i)^2 (-\mathbf{w}^T \mathbf{x}_i)$$

This additional terms pulls (or pushes) *proportionally*.
Let us now rewrite this J as sum of two parts one over $\mathcal{E}$ and the other on not in $\mathcal{E}$.

$$J = J_1 + J_2 = \sum_{\mathbf{x_i} \in \mathcal{E}} (t_i - o_i)^2 (-\mathbf{w}^T \mathbf{x}_i) + \sum_{\mathbf{x}_i \, not \in \mathcal{E}} (t_i - o_i)^2 (-\mathbf{w}^T \mathbf{x}_i)$$

$$J = J_1 + J_2 = \sum_{\mathbf{x_i} \in \mathcal{E}} (2 \cdot t_i)^2 (-\mathbf{w}^T \mathbf{x}_i) + \sum_{\mathbf{x}_i \, not \in \mathcal{E}} 0 \times (\mathbf{w}^T \mathbf{x}_i)$$

We know that J_2 is zero. When $\mathbf{x}_i \in \mathcal{E}$, $(t_i - o_i)$ is $2t_i$ i.e., $2y_i$.

Are the perceptrons doing GD?

- We know either $(t_i - o_i)$ is zero Or we know that $(t_i - o_i)$ is $2t_i$
- And $\frac{\partial(-\mathbf{w}^T\mathbf{x}_i)}{\partial \mathbf{w}} = -\mathbf{x_i}$.

$$\nabla J = -2\sum_{i=1}^{N}(t_i - o_i)\mathbf{x}_i$$

This leads to: $\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k - \eta\nabla J$ as:

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k - \eta\sum_{i=1}^{N}(t_i - o_i)(-\mathbf{x}_i)$$

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k + \eta\sum_{\mathbf{x}\in\mathcal{E}} 2.t_i\mathbf{x}_i$$

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k + \eta'\sum_{\mathbf{x}_i\in\mathcal{E}} y_i\mathbf{x}_i$$

(with changes in scale for learning rate η)

Perceptron Convergence

- **Claim 1** If there exist a set of weights that are consistent with the data, the perceptron algorithm will converge.
- **Claim 2**If the training data is not Linearly Separable, the perceptron algorithm will eventually repeat the same set of weights and thereby enter an infinite loop.
- **Claim 3** If the training data is linearly separable, algorithm will converge in a maximum of N steps. Find N.
- **Claim 4** Every boolean function can be represented by some network of perceptrons only two levels deep.

Read and understand the proofs for the above claims[1]

[1]Page 229 of Duda, Hart and Stork:
https://cds.cern.ch/record/683166/files/0471056693_TOC.pdf

Logistic Regression (LR)

Consider the classification with the $y \in \{-1, +1\}$ convention.

$$p(y = +1|\mathbf{x}) = g(\mathbf{w}^T\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}}$$

$$p(y = -1|\mathbf{x}) = 1 - g(\mathbf{w}^T\mathbf{x}) = \frac{1}{1 + e^{\mathbf{w}^T\mathbf{x}}}$$

We can combine both into single expression as

$$p(y_i|\mathbf{x_i}) = \frac{1}{1 + e^{-y_i\mathbf{w}^T\mathbf{x}_i}}$$

MLE based Loss Function

LR uses MLE based Loss. Likelihood (assuming independence) of the data is given by:

$$l(\mathbf{w}) = \prod_{i=1}^{N} p(y_i|\mathbf{x_i}; \mathbf{w})$$

We look for **w** that maximizes the likelihood

$$\mathbf{w}_{MLE} = \arg\max_{\mathbf{w}} \; l(\mathbf{w}) = \arg\max_{\mathbf{w}} \; \prod_{i=1}^{N} p(y_i|\mathbf{x_i}; \mathbf{w}) = \arg\max_{\mathbf{w}} \; \prod_{i=1}^{N} \frac{1}{1 + e^{-y_i \mathbf{w}^T \mathbf{x_i}}}$$

Take negative log likelihood:

$$\arg\min_{\mathbf{w}} \sum_{i=1}^{N} \log(1 + e^{-y_i \mathbf{w}^T \mathbf{x_i}})$$

SVM vs LR

Let us write the objective function corresponding to both SVM and Logistic regression schemes in a very similar manner.

SVM:

$$\min_{\mathbf{w}} C \sum_{i=1}^{N} \max(0, 1 - y_i f(\mathbf{x_i})) + ||\mathbf{w}||^2$$

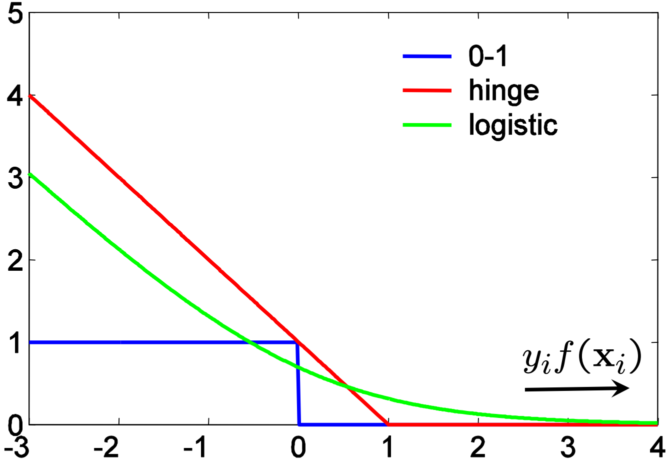
Logistic Regression

$$\min_{\mathbf{w}} \sum_{i=1}^{N} \log(1 + e^{-y_i f(\mathbf{x_i})}) + \lambda ||\mathbf{w}||^2$$

Both objective functions balance the relative importances with an additional term C VS λ. Parameters like this balance the relative importance of terms in an objective function. You may want to guess them right. But not very difficult in most cases.

Loss

Let us plot and see how different loss functions look like in the figure. You can see that both SVM and LR have very similar loss functions. One more smooth than the other.



Multiclass and Softmax

We know that:

$$p(y = 1|\mathbf{x}; \mathbf{w}) = \frac{e^{\mathbf{w}^T\mathbf{x}}}{1 + e^{\mathbf{w}^T\mathbf{x}}}$$

$$p(y = 0|\mathbf{x}; \mathbf{w}) = 1 - p(y = 1|\mathbf{x}; \mathbf{w}) = \frac{1}{1 + e^{\mathbf{w}^T\mathbf{x}}} = \frac{e^{-\mathbf{w}^T\mathbf{x}}}{1 + e^{-\mathbf{w}^T\mathbf{x}}}$$

Numerator is the weight/confidence of the sample being that class and denominator is the sum of weights for all classes. This allows us to extend to the multiclass setting as

$$p(y = c|\mathbf{x}; \mathbf{w_1}, \mathbf{w_2}, \mathbf{w_K}) = \frac{e^{\mathbf{w_c}^T\mathbf{x}}}{\sum_{i=1}^{K} e^{\mathbf{w_i}^T\mathbf{x}}}$$

Note that the sum of probabilities across all classes sum upto one. Finally a sample is classified into

$$\arg\max_i p(y = i|\mathbf{x})$$

This is also called popularly as **softmax**. Very popular in the modern deep learning architectures.

Perceptron: Problem

Consider two samples from A: $[1,2]^T, [4,2]^T$ and one sample from B: $[3,0]^T$. Let us start with a $\mathbf{w} = [1,1,0]^T$. Show one iteration of perceptron algorithm.