The Perceptron

Lecture Notes Consolidated with Claude AI — CS6140 Machine Learning

---

This note covers exactly what **HW2 Problem 3** needs. A companion note, **GD_LOGREG**, covers the same binary classification problem from a probabilistic angle (logistic regression); this one is entirely geometric, and older.

1 Setup

Same classification problem as logistic regression, but a different convention: labels are $y \in \{-1, +1\}$ (not $\{0, 1\}$ — a genuine, common convention change, not a typo). As usual, a dummy feature $x^0 = 1$ folds the intercept into the score, so $\mathbf{x} = (1, x^1, \ldots, x^m)$ and $\mathbf{w} = (w^0, w^1, \ldots, w^m)$. The predictor is

$$h_w(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x}) \in \{-1, +1\}.$$

2 Geometric Intuition

Instead of a probabilistic model, the perceptron directly searches for a hyperplane that separates the two classes.

2.1 The Hyperplane and Its Normal

The set $H = \{\mathbf{x} : \mathbf{w}^T\mathbf{x} = 0\}$ is a hyperplane through the origin (of the augmented feature space, where $x^0 = 1$ is just another coordinate). $\mathbf{w}$ is its **normal vector**: for any two points $\mathbf{x}_1, \mathbf{x}_2 \in H$, $\mathbf{w}^T\mathbf{x}_1 = \mathbf{w}^T\mathbf{x}_2 = 0 \Rightarrow \mathbf{w}^T(\mathbf{x}_1 - \mathbf{x}_2) = 0$, so $\mathbf{w}$ is perpendicular to every direction lying inside H.

2.2 Signed Distance to the Hyperplane

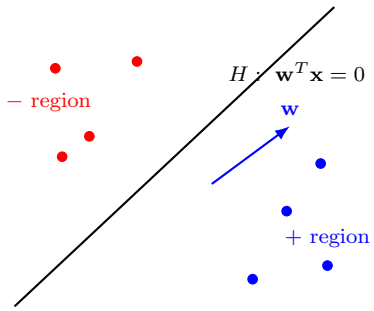
Write any point as $\mathbf{x} = \mathbf{x}_p + r\,\mathbf{w}/\|\mathbf{w}\|$, where $\mathbf{x}_p$ is $\mathbf{x}$'s projection onto H and r is what we want: the signed distance from $\mathbf{x}$ to H (positive on the side $\mathbf{w}$ points toward). Since $\mathbf{x}_p \in H$, $\mathbf{w}^T\mathbf{x}_p = 0$, so

$$\mathbf{w}^T\mathbf{x} = \mathbf{w}^T\mathbf{x}_p + r\,\frac{\mathbf{w}^T\mathbf{w}}{\|\mathbf{w}\|} = r\|\mathbf{w}\| \quad \Longrightarrow \quad r = \frac{\mathbf{w}^T\mathbf{x}}{\|\mathbf{w}\|}.$$

Key point: $\mathbf{w}^T\mathbf{x}$ is not just a sign to threshold — it is (up to the constant scale factor $\|\mathbf{w}\|$) the actual signed distance from $\mathbf{x}$ to the decision hyperplane. Points with $\mathbf{w}^T\mathbf{x} \gg 0$ are confidently on the "+" side, far from the boundary; points near $\mathbf{w}^T\mathbf{x} = 0$ are right on the fence. This is exactly the geometric picture that makes the update rule below (§4) sensible: it rotates the hyperplane using the misclassified point itself, i.e. using exactly the information about *which* side is wrong and by how far.

2.3 Linear Separability, Visualized

The whole enterprise only makes sense if such a hyperplane exists at all.



A linearly separable dataset: some hyperplane H (with normal $\mathbf{w}$) has every "+" point on one side and every "−" point on the other. §6 makes "how separable" precise via a margin γ.

3 A Useful Reformulation: The Sign-Flip Trick

The classification condition is $\mathbf{w}^T\mathbf{x}_i > 0$ if $y_i = +1$, and $\mathbf{w}^T\mathbf{x}_i < 0$ if $y_i = -1$ — two cases. A standard simplification: before training, replace every negative-class point $(\mathbf{x}_i, y_i{=}{-}1)$ with $(-\mathbf{x}_i, +1)$ (flip the sign of *both* the feature vector and the label). After this transformation, *every* point should satisfy the same single condition, $\mathbf{w}^T\tilde{\mathbf{x}}_i > 0$, regardless of its original class — there is no longer a case split. This is purely notational (it doesn't change the algorithm's behavior, and we won't use it below, preferring to keep y_i explicit), but it is how some textbooks and lecture slides present the perceptron, so it's worth recognizing.

4 The Objective and Its Gradient

A point is **misclassified** iff $y_i(\mathbf{w}^T\mathbf{x}_i) \le 0$ (predicted sign disagrees with the true label — equivalently, in §2's terms, $\mathbf{x}_i$ is on the wrong side, or exactly on the hyperplane). Let $M(w)$ be the set of currently misclassified points. The **perceptron criterion**:

$$J(w) = -\sum_{i \in M(w)} y_i(\mathbf{w}^T\mathbf{x}_i) \;\ge\; 0, \tag{1}$$

each term non-negative because $y_i(\mathbf{w}^T\mathbf{x}_i) \le 0$ for $i \in M(w)$; $J(w) = 0$ exactly when nothing is misclassified. Treating $M(w)$ as locally fixed, $\nabla_w J(w) = -\sum_{i \in M(w)} y_i\mathbf{x}_i$, giving the **batch** update rule (descending J):

$$\mathbf{w} \leftarrow \mathbf{w} + \eta \sum_{i \in M(w)} y_i\mathbf{x}_i, \tag{2}$$

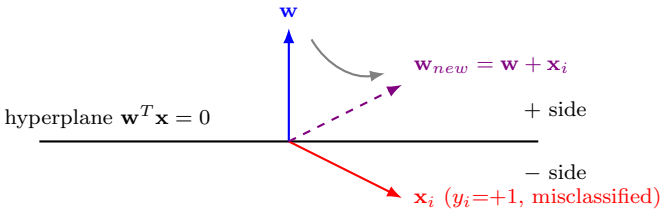
or, updating immediately on a single mistake $(\mathbf{x}_i, y_i)$ as soon as one is found (the usual formulation, and what HW2 Problem 3 asks for):

$$\mathbf{w} \leftarrow \mathbf{w} + \eta\, y_i\mathbf{x}_i. \tag{3}$$

Remark. Elsewhere (e.g. the summary table in **GD_LOGREG**) this same update is written $w^j \leftarrow w^j + \eta(y_i - h_w(\mathbf{x}_i))x_i^j$, "only on mistakes." At a mistake, $h_w(\mathbf{x}_i) = -y_i$ (the prediction disagrees), so $y_i - h_w(\mathbf{x}_i) = 2y_i$ — the same update as Equation 3 up to an overall factor of 2, absorbed into η. Both forms are standard; we derive with the $y_i\mathbf{x}_i$ form here since it matches the classical convergence proof below.

4.1 Geometric Picture of One Update

Adding $y_i\mathbf{x}_i$ to $\mathbf{w}$ rotates the hyperplane's normal *toward* $y_i\mathbf{x}_i$ — i.e. toward correctly classifying $\mathbf{x}_i$ next time.



$\mathbf{x}_i$ has true label $+1$ but sits on the hyperplane's $-$ side (a mistake). Adding $\mathbf{x}_i$ to $\mathbf{w}$ rotates the normal toward $\mathbf{x}_i$, moving the hyperplane so that $\mathbf{x}_i$ ends up on the $+$ side next time.

Algorithm: Perceptron (mistake-driven)

1. Initialize $\mathbf{w} = \mathbf{0}$.
2. Repeat until a full pass makes no mistakes: for each training point $(\mathbf{x}_i, y_i)$, if $y_i(\mathbf{w}^T\mathbf{x}_i) \le 0$ (a mistake), update $\mathbf{w} \leftarrow \mathbf{w} + \eta\, y_i\mathbf{x}_i$.

---

5 Worked Example: Learning the Boolean AND Function

Four points, $\mathbf{x} = (x^0{=}1, x^1, x^2)$, labels $y = \text{AND}(x^1, x^2)$ remapped from $\{0,1\}$ to $\{-1,+1\}$:

i	x^0	x^1	x^2	y_i
1	1	0	0	-1
2	1	0	1	-1
3	1	1	0	-1
4	1	1	1	$+1$

Start $\mathbf{w} = (0,0,0)$, $\eta = 1$, process points in order $1,2,3,4$ (one pass = one **epoch**):

- i=1: $\mathbf{w}^T\mathbf{x}_1 = 0$; $y_1 \cdot 0 = 0 \le 0$, a mistake. Update: $\mathbf{w} \leftarrow (0,0,0) + (-1)(1,0,0) = (-1,0,0)$.
- i=2: $\mathbf{w}^T\mathbf{x}_2 = (-1,0,0)\cdot(1,0,1) = -1$; $y_2 \cdot (-1) = (-1)(-1) = 1 > 0$, correct, no update.
- i=3: $\mathbf{w}^T\mathbf{x}_3 = (-1,0,0)\cdot(1,1,0) = -1$; $y_3 \cdot (-1) = 1 > 0$, correct, no update.
- i=4: $\mathbf{w}^T\mathbf{x}_4 = (-1,0,0)\cdot(1,1,1) = -1$; $y_4 \cdot (-1) = (+1)(-1) = -1 \le 0$, a mistake. Update: $\mathbf{w} \leftarrow (-1,0,0) + (1)(1,1,1) = (0,1,1)$.

After one epoch, $\mathbf{w} = (0,1,1)$ — *not yet* separating: $\mathbf{w}^T\mathbf{x}_1 = 0$ (a tie, still counted as a mistake since it isn't < 0), $\mathbf{w}^T\mathbf{x}_2 = 1$ and $\mathbf{w}^T\mathbf{x}_3 = 1$ (both wrongly positive, true label -1), only $\mathbf{x}_4$ correct. Three of four points still need fixing, so the algorithm continues for further epochs (full trace omitted). It eventually converges to some separating $\mathbf{w}^*$ — for instance $\mathbf{w}^* = (-4,3,2)$ works: $\mathbf{w}^{*T}\mathbf{x}_1 = -4 < 0$, $\mathbf{x}_2$: $-4+2 = -2 < 0$, $\mathbf{x}_3$: $-4+3 = -1 < 0$, $\mathbf{x}_4$: $-4+3+2 = 1 > 0$ — all four strictly correct.

Key point: This is a genuine, hand-traceable run of exactly the algorithm HW2 Problem 3 asks you to implement — useful as a unit test: feed your implementation this same 4-point dataset and $\mathbf{w}_0 = \mathbf{0}$, and check your first-epoch trace matches the one above exactly before trusting it on the real (Perceptron) dataset.

6 Convergence: The Mistake Bound

Assume the data is **linearly separable with margin** γ: there exists a unit vector $\bar{\mathbf{w}}$ ($\|\bar{\mathbf{w}}\| = 1$) and $\gamma > 0$ with $y_i(\bar{\mathbf{w}}^T \mathbf{x}_i) \geq \gamma$ for every training point i (§2's Figure). Assume also the data is bounded, $\|\mathbf{x}_i\| \leq R$ for all i. Start from $\mathbf{w}_0 = \mathbf{0}$; let $\mathbf{w}_k$ be the weight vector after the k-th mistake (using $\eta = 1$, without loss of generality).

Claim 1: $\mathbf{w}_k \cdot \bar{\mathbf{w}} \geq k\gamma$. *Proof by induction*: true at $k = 0$ ($0 \geq 0$). If it holds at k, then for the mistake $(\mathbf{x}_k, y_k)$ that produced $\mathbf{w}_{k+1} = \mathbf{w}_k + y_k \mathbf{x}_k$:

$$\mathbf{w}_{k+1} \cdot \bar{\mathbf{w}} = \mathbf{w}_k \cdot \bar{\mathbf{w}} + y_k(\mathbf{x}_k \cdot \bar{\mathbf{w}}) \geq k\gamma + \gamma = (k+1)\gamma,$$

using the margin assumption on the second term.

Claim 2: $\|\mathbf{w}_k\|^2 \leq kR^2$. *Proof by induction*: true at $k = 0$. If it holds at k:

$$\|\mathbf{w}_{k+1}\|^2 = \|\mathbf{w}_k\|^2 + 2y_k(\mathbf{w}_k \cdot \mathbf{x}_k) + \underbrace{y_k^2}_{=1} \|\mathbf{x}_k\|^2 \;\leq\; \|\mathbf{w}_k\|^2 + \|\mathbf{x}_k\|^2 \;\leq\; kR^2 + R^2 = (k+1)R^2,$$

since $(\mathbf{x}_k, y_k)$ was a mistake, $y_k(\mathbf{w}_k \cdot \mathbf{x}_k) \leq 0$.

Combine via Cauchy–Schwarz ($\|\bar{\mathbf{w}}\| = 1$): $k\gamma \leq \mathbf{w}_k \cdot \bar{\mathbf{w}} \leq \|\mathbf{w}_k\|\|\bar{\mathbf{w}}\| = \|\mathbf{w}_k\| \leq R\sqrt{k}$. So $k\gamma \leq R\sqrt{k} \Rightarrow \sqrt{k} \leq R/\gamma$, i.e.

$$k \;\leq\; \left(\frac{R}{\gamma}\right)^2. \tag{4}$$

Key point: The perceptron makes at most $(R/\gamma)^2$ mistakes *total*, ever — not per epoch, in its entire run — and therefore terminates in finite time whenever the data is linearly separable (**HW2 Problem 3**'s guarantee that your mistake count reaches 0). The bound only depends on how well-separated the data is (γ, larger is easier) and how spread out it is (R, smaller is easier) — not on the number of features or the number of training points directly.

7 Implementation Notes (HW2)

- **Problem 3** (perceptron from scratch): Equation 3, looping over the training set until a full pass finds no mistakes (guaranteed by §6, since the given dataset is linearly separable). Sanity-check your implementation against §5's worked trace before running it on the real data. Library check: `sklearn`'s `Perceptron`.
- **Common bug**: forgetting the dummy feature $x^0 = 1$, which silently forces the separating hyperplane through the origin (no learnable intercept) — fine only if the data happens to be separable through the origin already, which real datasets rarely are.