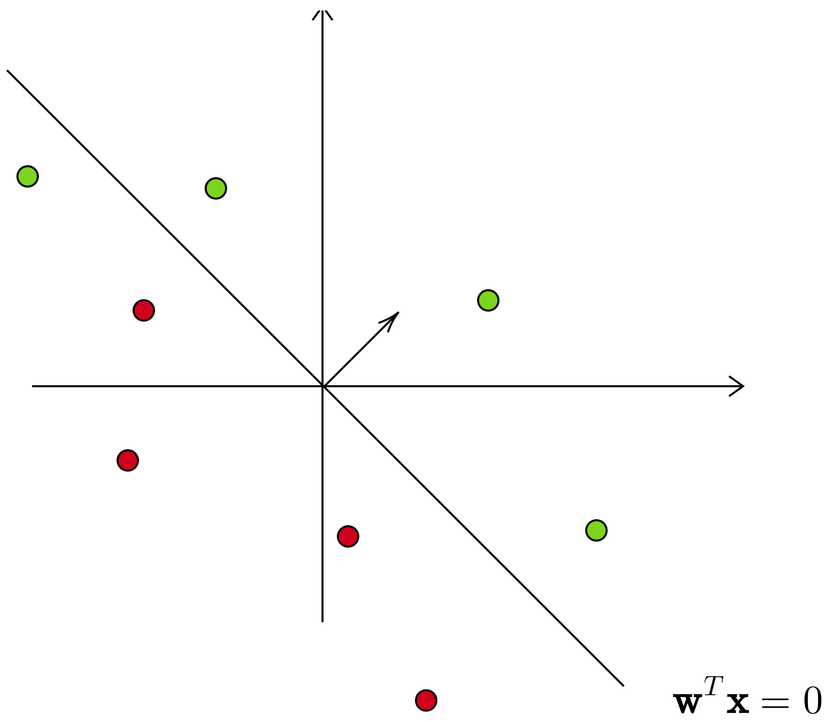
Linear Classifiers

Perceptron

Perceptron is a simple discriminative model that works on linearly separable data:



The labels for a perceptron model lie in the set $\{-1, 1\}$.

Model

$$P(y = 1 \mid \mathbf{x}) = \begin{cases} 1, & \mathbf{w}^T\mathbf{x} \geqslant 0 \\ 0, & \mathbf{w}^T\mathbf{x} < 0 \end{cases}$$

Prediction

The prediction for any data-point $\mathbf{x}$ and weight vector $\mathbf{w}$ is given as follows:

$$\widehat{y} = \begin{cases} 1, & \mathbf{w}^T\mathbf{x} \geqslant 0 \\ -1, & \mathbf{w}^T\mathbf{x} < 0 \end{cases}$$

The decision boundary is a hyperplane in $\mathbb{R}^d$.

Perceptron learning algorithm:

(1) Start with $\mathbf{w}^0 = 0$

(2) Cycle through the n data-points in a particular order:

    (3) For $\mathbf{w}^t$, if $(\mathbf{x}_i, y_i)$ is a mistake, then update:

$$\mathbf{w}^{t+1} := \mathbf{w}^t + \mathbf{x}_i \cdot y_i$$

    (4) If all points are correctly classified, exit the loop

Types of mistakes

(1) False positives: $y_i = -1$, $\widehat{y}_i = 1$

(2) False negatives: $y_i = 1$, $\widehat{y}_i = -1$

Convergence

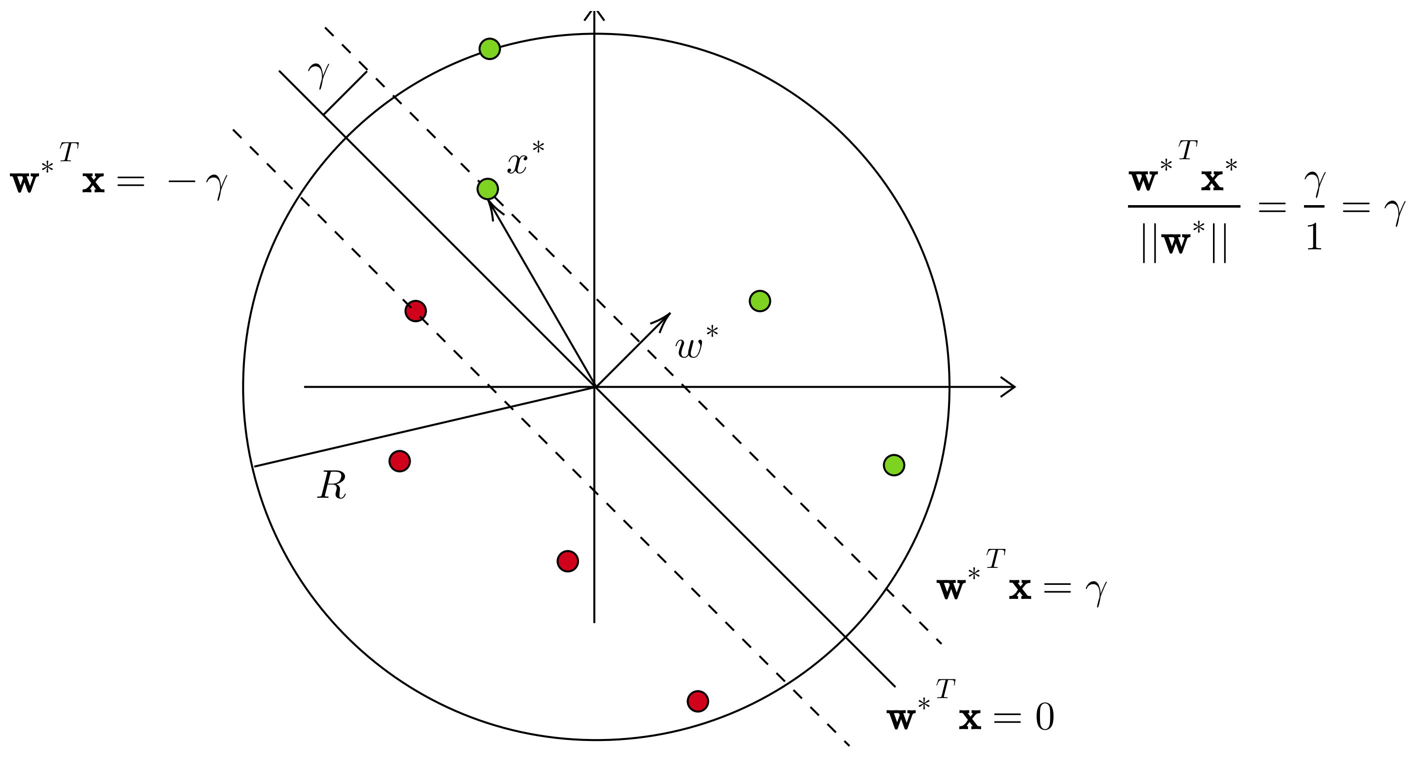
Perceptron algorithm converges for a linearly separable dataset with γ-margin:

(1) Radius assumption: There exists some $R > 0$ such that:

$$||\mathbf{x}_i||^2 \leqslant R^2, \quad 1 \leqslant i \leqslant n$$

(2) Linear separability with γ-margin: There exists some $\mathbf{w}^*$ with $||\mathbf{w}^*|| = 1$ and some $\gamma > 0$ such that:

$$\left(\mathbf{w}^{*T}\mathbf{x}_i\right) y_i \geqslant \gamma, \quad 1 \leqslant i \leqslant n$$


γ
${\mathbf{w}^*}^T\mathbf{x} = -\gamma$
x^*
w^*
R
$\frac{{\mathbf{w}^*}^T\mathbf{x}^*}{||\mathbf{w}^*||} = \frac{\gamma}{1} = \gamma$
${\mathbf{w}^*}^T\mathbf{x} = \gamma$
${\mathbf{w}^*}^T\mathbf{x} = 0$

For a linearly separable dataset with γ-margin, perceptron converges. It makes at most $\frac{R^2}{\gamma^2}$ mistakes in the process. In other words, the weight vector is updated at most $\frac{R^2}{\gamma^2}$ times in the algorithm.

<u>Remark-1</u>

Since each iteration of the perceptron update rule adds some data-point to the weight vector, we have:

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i y_i$$

Here, α_i is the number of times the weight vector was updated using $(\mathbf{x}_i, y_i)$.

<u>Remark-2</u>

Loosely speaking, perceptron learning algorithm can be viewed as a case of stochastic gradient descent on the **modified hinge loss** with unit learning rate. The modified hinge loss for a single data-point is:

$$L(\mathbf{w}, \mathbf{x}_i, y_i) = \begin{cases} -\left(\mathbf{w}^T\mathbf{x}_i\right)y_i, & \left(\mathbf{w}^T\mathbf{x}_i\right)y_i < 0 \\ 0, & \left(\mathbf{w}^T\mathbf{x}_i\right)y_i \geqslant 0 \end{cases}$$

The update rule is:

$$\mathbf{w}^{t+1} := \mathbf{w}^t - \alpha \nabla L(\mathbf{w}, \mathbf{x}_i, y_i)$$

Crudely speaking, the "gradient" is $-\mathbf{x}_i y_i$ for a mistake and 0 for a correctly classified point. We can now recover the perceptron update rule:

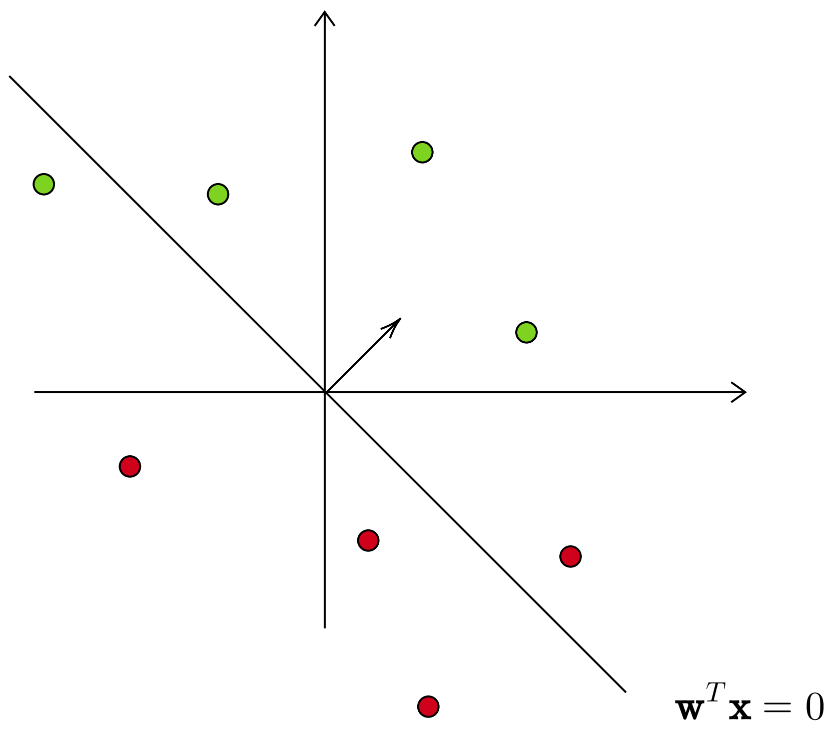
$$\mathbf{w}^{t+1} := \mathbf{w}^t - (1)(-\mathbf{x}_i y_i)$$

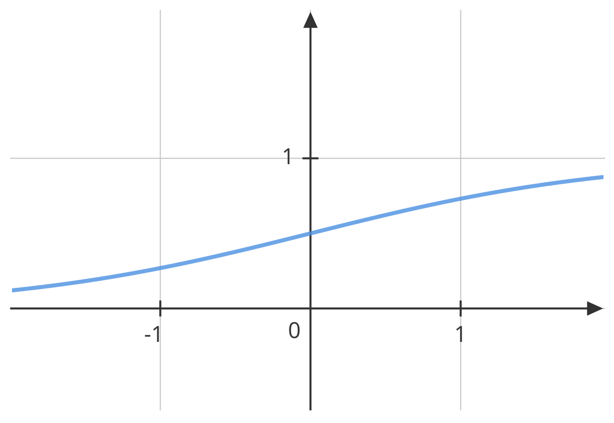
This becomes:

$$\mathbf{w}^{t+1} := \mathbf{w}^t + \mathbf{x}_i y_i$$

Logistic Regression

Logisitc regression is a discriminative model that models the conditional probability of label given the data-point.





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

Model

$$P(y=1 \mid \mathbf{x}) = \sigma\left(\mathbf{w}^T\mathbf{x}\right) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$$

Prediction

For a threshold T, the prediction is:

$$\widehat{y} = \begin{cases} 1, & P(y=1 \mid \mathbf{x}) \geqslant T \\ 0, & P(y=1 \mid \mathbf{x}) < T \end{cases}$$

Likelihood

The likelihood of observing the data for a weight vector $\mathbf{w}$:

$$L(\mathbf{w}) = \prod_{i=1}^{n} \sigma\left(\mathbf{w}^T\mathbf{x}_i\right)^{y_i} \cdot \left[1 - \sigma\left(\mathbf{w}^T\mathbf{x}_i\right)\right]^{1-y_i}$$

- If $y_i = 1$, we want p_i to be close to 1.
- If $y_i = 0$, we want p_i to be close to 0.

In simple terms, we want p_i to be close to y_i.

Loss

The loss for logistic regression is the negative log-likelihood, also called the binary cross-entropy loss:

$$l(\mathbf{w}) = \sum_{i=1}^{n} -y_i \cdot \log\Big[\sigma\Big(\mathbf{w}^T\mathbf{x}_i\Big)\Big] - (1-y_i) \cdot \log\Big[1-\sigma\Big(\mathbf{w}^T\mathbf{x}_i\Big)\Big]$$

The binary cross-entropy loss is a convex function of the weights. Hence, it can be optimized using gradient descent. With an appropriate choice of the learning rate, gradient descent will converge to the global minimum of the loss function.

Gradient Descent

Gradient of the loss:

$$\begin{aligned}\nabla l(\mathbf{w}) &= \sum_{i=1}^{n} -y_i \cdot \Big[1-\sigma\Big(\mathbf{w}^T\mathbf{x}_i\Big)\Big]\mathbf{x}_i + (1-y_i)\sigma\Big(\mathbf{w}^T\mathbf{x}_i\Big)\mathbf{x}_i \\ &= \sum_{i=1}^{n} \Big[\sigma\Big(\mathbf{w}^T\mathbf{x}_i\Big) - y_i\Big] \cdot \mathbf{x}_i \\ &= \sum_{i=1}^{n} [p_i - y_i] \cdot \mathbf{x}_i \\ &= \sum_{i=1}^{n} e_i \cdot \mathbf{x}_i\end{aligned}$$

Gradient for a single data-point can be viewed as:

$$\texttt{gradient} = \texttt{error} \times \texttt{data-point}$$

Smaller the error for a data-point, smaller its contribution to the update. Initialize $\mathbf{w}^0 = 0$. One step of gradient descent:

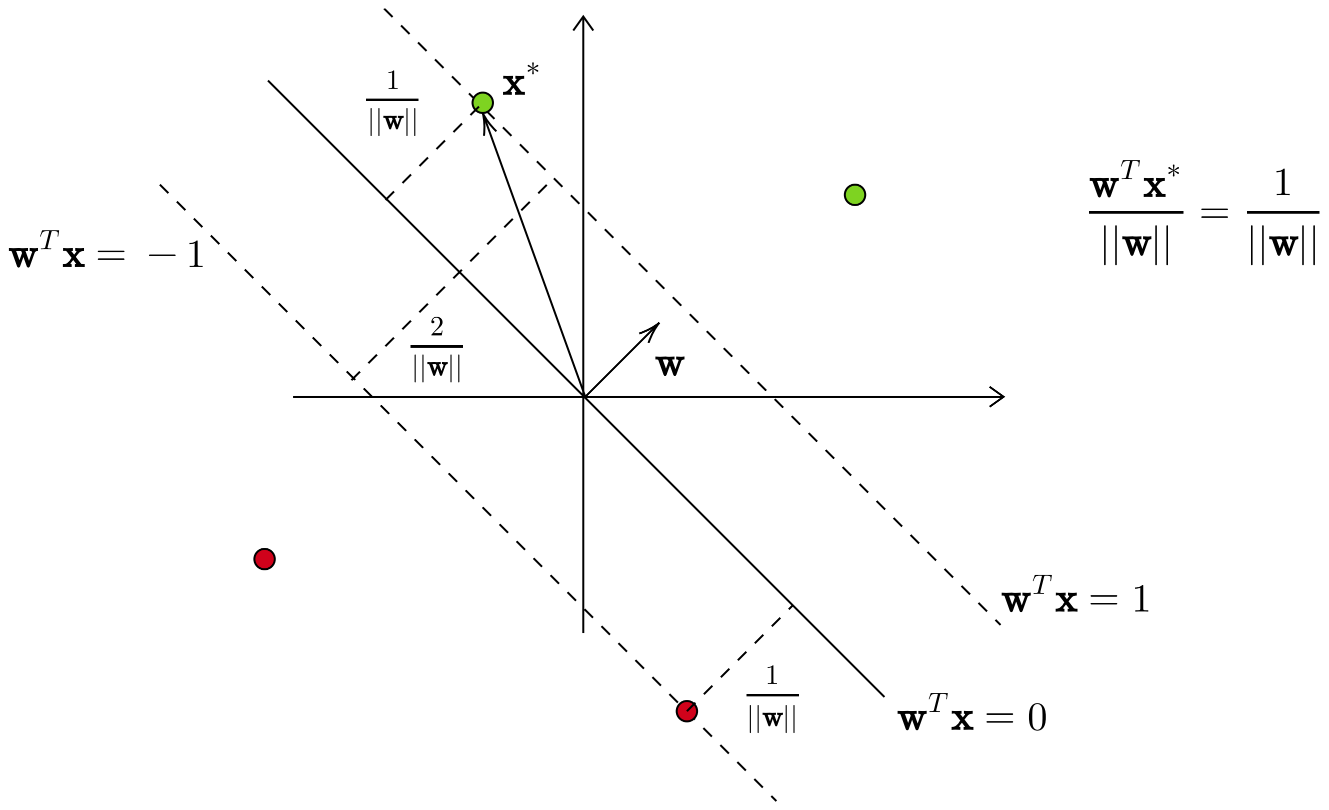
$$\mathbf{w}^{t+1} := \mathbf{w}^{t} - \alpha \cdot \nabla l(\mathbf{w}^{t})$$

Remark

Note that logistic regression is not constrained by linear separability and returns an optimal weight vector even if the data is not linearly separable.

SVM

We look at hard-margin, linear-SVM for a linearly separable dataset. The margin is the distance of the nearest point to the boundary. If the closest point lies on $\mathbf{w}^T\mathbf{x} = \pm 1$, then the margin turns out to be $\dfrac{1}{||\mathbf{w}||}$.



$\frac{1}{||\mathbf{w}||}$
$\mathbf{x}^*$
$\dfrac{\mathbf{w}^T\mathbf{x}^*}{||\mathbf{w}||} = \dfrac{1}{||\mathbf{w}||}$
$\mathbf{w}^T\mathbf{x} = -1$
$\frac{2}{||\mathbf{w}||}$
$\mathbf{w}$
$\mathbf{w}^T\mathbf{x} = 1$
$\frac{1}{||\mathbf{w}||}$
$\mathbf{w}^T\mathbf{x} = 0$

Primal

Goal is to maximize the margin subject to the points being beyond the margin on the correct side of the boundary:

$$\min_{w} \quad \frac{||\mathbf{w}||^2}{2}$$

sub. to

$$\left(\mathbf{w}^T\mathbf{x}_i\right)y_i \geqslant 1, \quad 1 \leqslant i \leqslant n$$

Dual

The Lagrangian is:

$$L(\mathbf{w}, \boldsymbol{\alpha}) = \frac{||\mathbf{w}||^2}{2} + \sum_{i=1}^{n} \alpha_i \left[1 - \left(\mathbf{w}^T\mathbf{x}_i\right)y_i\right]$$

We can now write the dual as:

$$\max_{\boldsymbol{\alpha}\geqslant 0} \;\; \min_{\mathbf{w}} \;\; \frac{||\mathbf{w}||^2}{2} + \sum_{i=1}^{n} \alpha_i \Big[1 - \big(\mathbf{w}^T \mathbf{x}_i\big) y_i\Big]$$

The inner minimization problem is unconstrained. Therefore:

$$\nabla_{\mathbf{w}} L(\mathbf{w}, \boldsymbol{\alpha}) = 0 \Longrightarrow \mathbf{w} = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i y_i$$

In matrix form:

$$\mathbf{w} = \mathbf{X}\mathbf{Y}\boldsymbol{\alpha}$$

With this, the dual becomes a maximization problem over $\boldsymbol{\alpha}$:

$$\max_{\boldsymbol{\alpha}\geqslant 0} \quad \boldsymbol{\alpha}^T 1 - \frac{1}{2} \cdot \boldsymbol{\alpha}^T \big(\mathbf{Y}^T \mathbf{X}^T \mathbf{X}\mathbf{Y}\big) \boldsymbol{\alpha}$$

One way to remember the dual is by introducing a matrix $\mathbf{Q}$:

$$\mathbf{Q} = (\mathbf{X}\mathbf{Y})^T (\mathbf{X}\mathbf{Y})$$

The dual therefore becomes:

$$\max_{\boldsymbol{\alpha}\geqslant 0} \quad \boldsymbol{\alpha}^T \mathbf{1} - \frac{1}{2} \cdot \boldsymbol{\alpha}^T \mathbf{Q} \boldsymbol{\alpha}$$

The dual with vectors/matrices replaced by scalars:

$$\max_{\boldsymbol{\alpha}\geqslant 0} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \cdot \sum_{i=1}^{n} \sum_{j=1}^{n} Q_{ij} \alpha_i \alpha_j$$

The dual is a concave, quadratic programming problem.

<u>KKT Conditions (optimality)</u>

If $\mathbf{w}^*$ is the primal optimal solution and $\boldsymbol{\alpha}^*$ is the dual optimal solution, then the following conditions hold:

(1) Stationarity of the Lagrangian:

$$\mathbf{w}^* = \sum_{i=1}^{n} \alpha_i^* \mathbf{x}_i y_i$$

(2) Complementary slackness conditions:

$$\alpha_i^* \cdot \left[1 - \left({\mathbf{w}^*}^T \mathbf{x}_i\right) y_i\right] = 0, \quad 1 \leqslant i \leqslant n$$

(3) Primal feasibility:

$$\left({\mathbf{w}^*}^T \mathbf{x}_i\right) y_i \geqslant 1, \quad 1 \leqslant i \leqslant n$$

(4) Dual feasibility

$$\alpha_i^* \geqslant 0, \ \ 1 \leqslant i \leqslant n$$

<u>Support vector</u>

A support vector is a point for which $\alpha_i^* > 0$. From complementary slackness, every support vector lies on the supporting hyperplanes:

$$\alpha_i^* > 0 \Longrightarrow 1 - \left({\mathbf{w}^*}^T \mathbf{x}_i\right) y_i = 0 \Longrightarrow \left({\mathbf{w}^*}^T \mathbf{x}_i\right) y_i = 1$$

SVM is a sparse linear combination of the data-points. The sparsity comes from the fact that $\alpha_i^* = 0$ for most of the data-points. Only the support vectors contribute or "support" in forming the decision boundary.

<u>Prediction</u>

For a test point, the predicted label is:

$$\text{sign}\left({\mathbf{w}^*}^T \mathbf{x}_{\text{test}}\right)$$

<u>Kernel SVM</u>

Non-linear decision boundaries can be learnt with the help of kernels. The dual for kernel-SVM:

$$\max_{\boldsymbol{\alpha} \geqslant 0} \quad \boldsymbol{\alpha}^T \mathbf{1} - \frac{1}{2} \cdot \boldsymbol{\alpha}^T \left(\mathbf{Y}^T \mathbf{K} \mathbf{Y}\right) \boldsymbol{\alpha}$$

The weight vector can't be computed explicitly. However, the predicted label can be computed as:

$$\text{sign}\left[\sum_{i=1}^{n} \alpha_i^* \cdot k(\mathbf{x}_i, \mathbf{x}_{\text{test}}) \cdot y_i\right]$$

The term $k(\mathbf{x}_i, \mathbf{x}_{\text{test}})$ can be seen as computing some kind of similarity score between $\mathbf{x}_i$ and $\mathbf{x}_{\text{test}}$. This value together with α_i^* gives the importance of $\mathbf{x}_i$ in predicting the label of $\mathbf{x}_{\text{test}}$.