Linear Models

CMPUT 261: Introduction to Artificial Intelligence

P&M §7.3

Assignment #2

Assignment #2 is due **Oct 17/2023** (one week from today) at **11:59pm**

- Submissions past the deadline will have late penalty applied
- Leave yourself some margin for error when submitting!

Recap: Supervised Learning

Definition: A supervised learning task consists of

- A set of **input features** $X_1, \ldots, X_n$
- A set of **target features** $Y_1, \ldots, Y_k$
- A set of **training examples**, for which both input and target features are given
- A set of **test examples**, for which only the input features are given

The goal is to **predict** the values of the **target features** given the **input features**; i.e., **learn** a function $h(x)$ that will map features X to a prediction of Y

- We want to predict **new, unseen data** well; this is called **generalization**
- Can **estimate generalization** performance by reserving separate **test examples**

Recap: Loss Functions

- A **loss function** gives a quantitative measure of a hypothesis's performance
- There are many commonly-used loss functions, each with its own properties

Loss	Definition
0/1 error	$\sum_{e \in E} 1 \left[Y(e) \neq \hat{Y}(e) \right]$
absolute error	$\sum_{e \in E} \left\lvert Y(e) - \hat{Y}(e) \right\rvert .$
squared error	$\sum_{e \in E} \left(Y(e) - \hat{Y}(e) \right)^2 .$
worst case	$\max_{e \in E} \left\lvert Y(e) - \hat{Y}(e) \right\rvert .$
likelihood	$\Pr(E \mid \hat{Y}) = \prod_{e \in E} \hat{Y}(e = Y(e))$
log-likelihood	$\log \Pr(E \mid \hat{Y}) = \sum_{e \in E} \log \hat{Y}(e = Y(e)) .$

Recap: Optimal Trivial Predictors for Binary Data

- Suppose we are predicting a **binary** target
- n_0 **negative** examples
- n_1 **positive** examples
- **Question:** What is the optimal single prediction?

Measure	Optimal Prediction
0/1 error	0 if $n_0 > n_1$ else 1
absolute error	0 if $n_0 > n_1$ else 1
squared error	$\frac{n_1}{n_0 + n_1}$
worst case	$\begin{cases} 0 & \text{if } n_1 = 0 \\ 1 & \text{if } n_0 = 0 \\ 0.5 & \text{otherwise} \end{cases}$
likelihood	$\frac{n_1}{n_0 + n_1}$
log-likelihood	$\frac{n_1}{n_0 + n_1}$

Lecture Outline

1. Recap & Logistics
2. Linear Regression
3. Linear Classification

After this lecture, you should be able to:

- specify and/or implement linear regression, linear classification, logistic regression
- explain the benefits of different approaches to learning linear models

Linear Regression

- Linear regression is the problem of fitting a **linear function** to a set of training examples
  - Both input and target features must be **numeric**
- **Linear function** of the input features:

$$\hat{Y}^w(e) = w_0 + w_1 X_1(e) + \ldots + w_d X_d(e)$$
$$= \sum_{j=0}^{d} w_i X_i(e)$$

For convenience, we often add a special "constant feature" $X_0(e) = 1$ for all examples

Ordinary Least-Squares

For the squared error loss, it is possible to find the optimal predictor for a dataset **analytically**:

1. $L(w) = \sum_{e \in E} \left(Y(e) - \hat{Y}^w(e) \right)^2 = \sum_{e \in E} \left(Y(e) - \sum_{j=0}^{d} w_j X_j(e) \right)^2$

2. Recall that $\nabla L(w^*) = 0$ for $w^* \in \arg \min_{w \in \mathbb{R}^{d+1}} L(w)$

3. Derive an expression for $\nabla L(w^*)$ and solve for 0

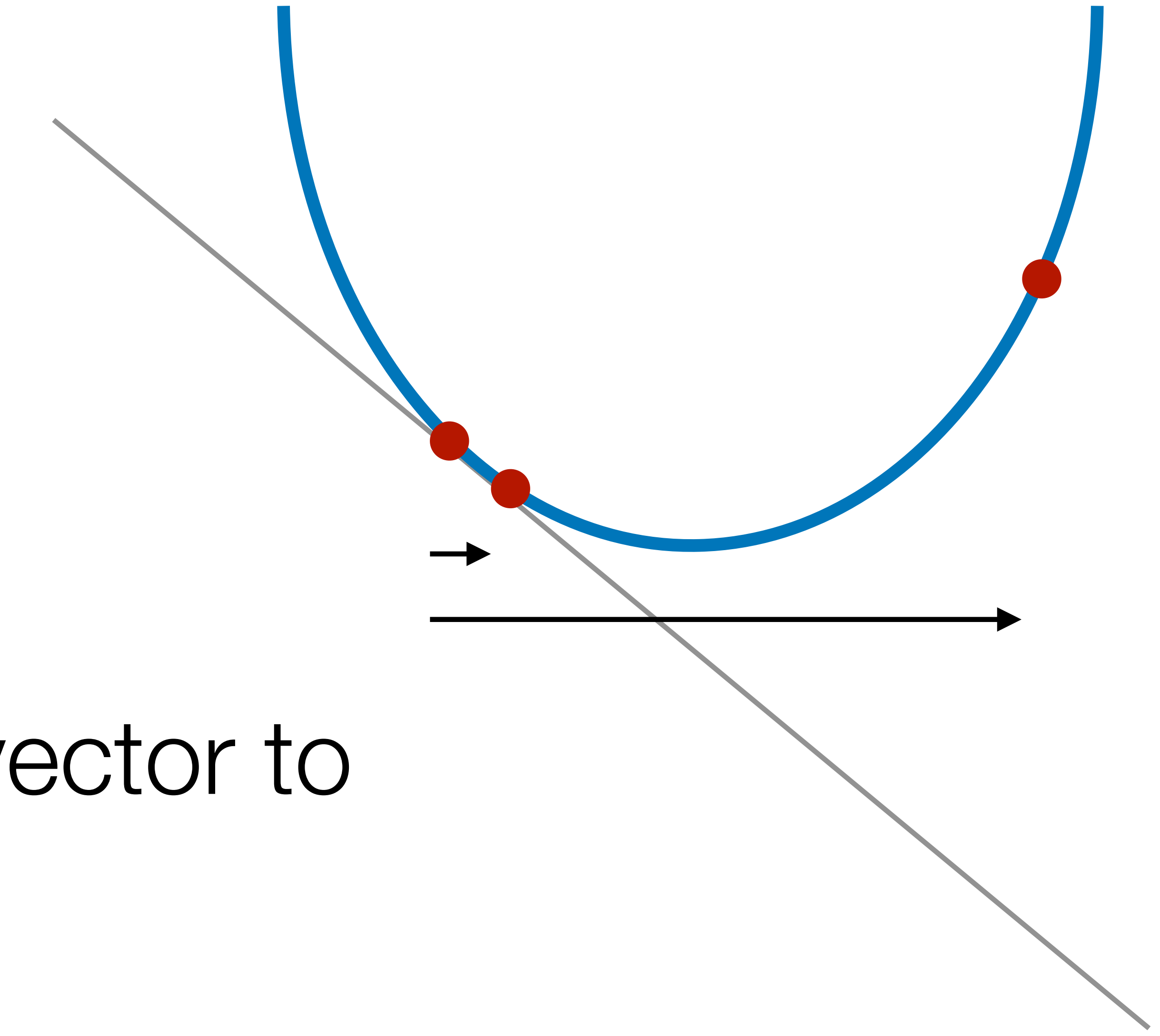
- For d input features, solve a system of $d + 1$ equations
- Requires inverting a $(d+1) \times (d+1)$ matrix $O(d^3)$
- Constructing the matrix requires adding n matrices (one for each example) $O(nd^2)$
- Total cost: $O(nd^2 + d^3)$

Gradient Descent

- The analytic solution is tractable for **small** datasets with **few** input features
  - ImageNet has about **14 million images** with $256 \times 256 = 65{,}536$ input features
- For others, we use **gradient descent**
  - Gradient descent is an iterative method to find the minimum of a function.
  - For **minimizing error**:

$$w_j^{(t+1)} \leftarrow w_j^{(t)} - \eta \frac{\partial}{\partial w_j^{(t)}} \mathrm{error}(E, w^{(t)})$$

Gradient Descent



- The gradient of a function tells how to change every element of a vector to **increase** the function
  - If the partial derivative of x_i is positive, increase x_i
- **Gradient descent:**
  Iteratively choose new values of x in the (opposite) direction of the gradient:

$$\mathbf{x}^{new} = \mathbf{x}^{old} - \eta \nabla f(\mathbf{x}^{old}) \, .$$

learning rate

  - This only works for **sufficiently small** changes (**why?**)
  - **Question:** How much should we change $\mathbf{x}^{old}$?

Gradient Descent Variations

- **Incremental gradient descent:** update each weight after **each example** in turn

$$\forall e_i \in E : w_j^{(t+1)} \leftarrow w_j^{(t)} - \eta \frac{\partial}{\partial w_j^{(t)}} \text{error}\left(\{e_i\}, w^{(t)}\right)$$

- **Batched gradient descent:** update each weight based on a **batch** of examples

$$\forall E_i : w_j^{(t+1)} \leftarrow w_j^{(t)} - \eta \frac{\partial}{\partial w_j^{(t)}} \text{error}\left(E_i, w^{(t)}\right)$$

- **Stochastic gradient descent:** update repeatedly on **random** examples:

$$e_i^t \sim U(E) : w_j^{(t+1)} \leftarrow w_j^{(t)} - \eta \frac{\partial}{\partial w_j^{(t)}} \text{error}\left(\{e^t\}, w^{(t)}\right)$$

Question

Why would we ever use any of these?

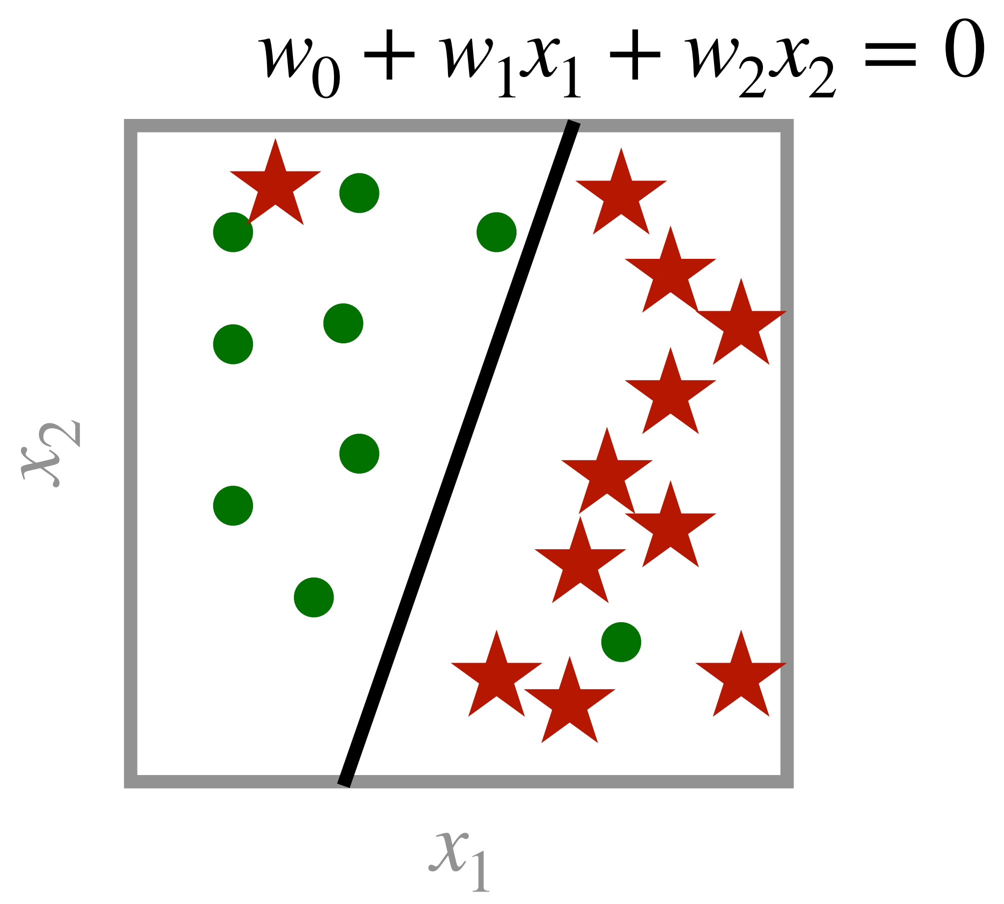
Linear Classification

- For **binary** targets, we can use linear regression to do classification
- Represent binary classes by $\{-1, +1\}$
- If regression target is negative, predict -1, else predict $+1$

$$\hat{Y}^w(e) = \mathrm{sgn}\left(\sum_{i=0}^{n} w_i X_i(e)\right)$$

sgn returns +1 for positive arguments and -1 for negative arguments

- The line defined by $\sum_{i=0}^{n} w_i x_i = 0$ is called the **decision boundary**



Probabilistic Linear Classification

- For **binary targets** represented by $\{0,1\}$ or **numeric input** features, we can use linear function to estimate the **probability** of the class
- **Issue:** we need to constrain the output to lie within $[0,1]$
- Instead of outputting results of the function directly, send it through an **activation function** $f : \mathbb{R} \to [0,1]$ instead:

$$\hat{Y}^w(e) = f\left(\sum_{i=0}^{n} w_i X_i(e)\right)$$

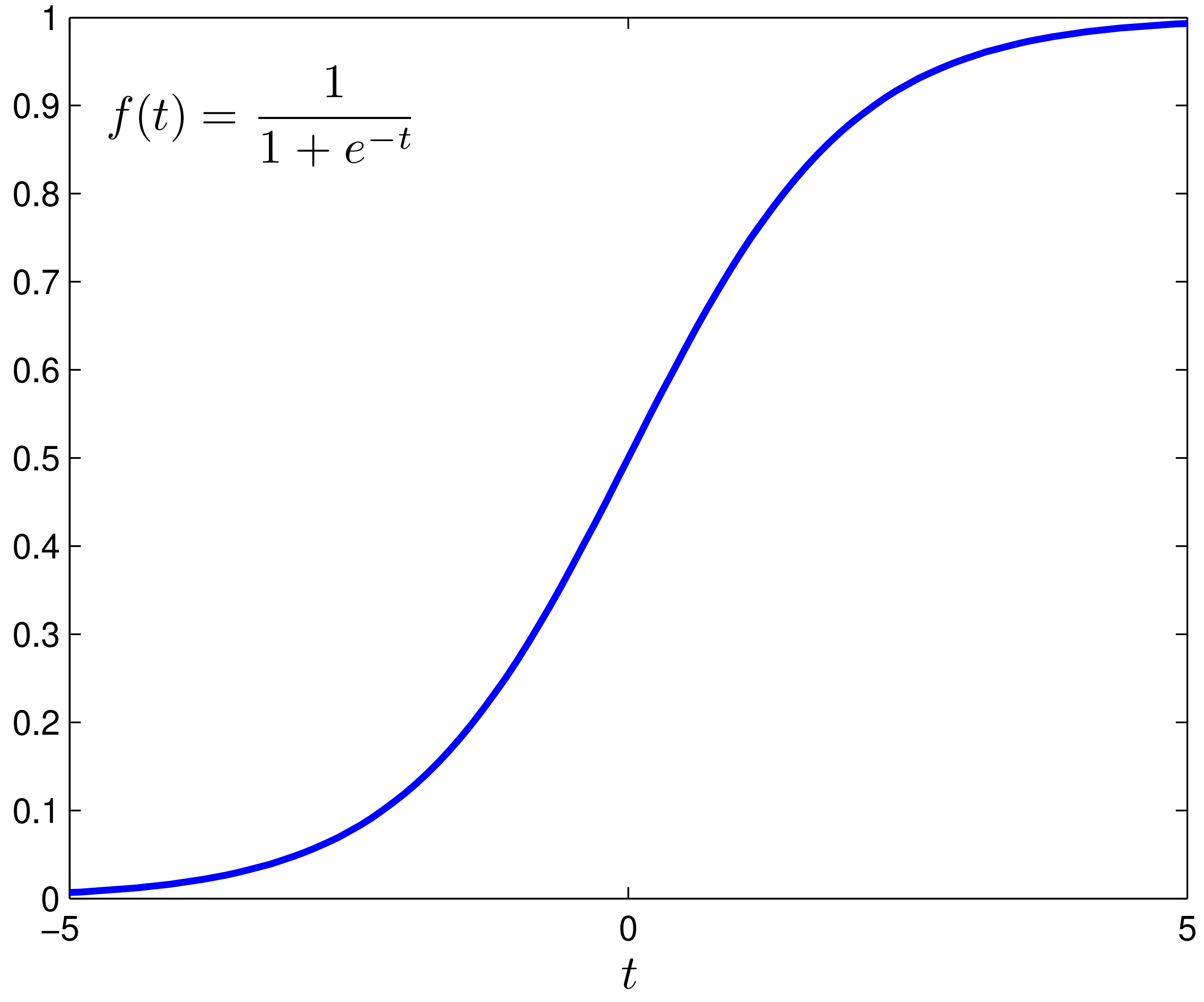
Logistic Regression

- A very commonly used activation function is the **logistic** function:

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

- Linear classification with a logistic activation function is often referred to as **logistic regression:**

$$\hat{Y}^w(e) = s\left(\sum_{i=0}^{n} w_i X_i(e)\right)$$



Question: What is the **decision boundary** in logistic regression?

Non-Binary Target Features

What if the target feature has $k > 2$ values?

1. Use k **indicator** variables
2. Learn each indicator variable **separately**
3. **Normalize** the predictions:

$$\hat{Y}^{w}_{m}(e) = \frac{e^{\left(\sum_{j=0}^{d} w_{m,j} X_j(e)\right)}}{\sum_{\ell=1}^{k} e^{\left(\sum_{j=0}^{d} w_{\ell,j} X_i(e)\right)}}$$

Summary

- **Linear regression** is a simple model for predicting real quantities
- **Linear classification** can be built from linear regression
  - Based on **sign** of prediction ("discriminative"), or
  - Using **logistic regression** ("probabilistic")
  - For **non-binary target features**, can normalize probabilistic predictions for individual classes
- **Gradient descent** is a general, widely-used training procedure (with several variants)
  - Linear models can be optimized in **closed form** for certain losses
  - In practice often optimized with gradient descent