Linear Models

CMPUT 261: Introduction to Artificial Intelligence

P&M §7.3

Assignment #2

Assignment #2 is due **Thu Feb 29/2024** (two weeks from today) at **11:59pm**

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

Next week is **reading week**

- No lectures or labs next week

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	$\sum_{i=1}^{n} 1\left[y^{(i)} \neq h(\mathbf{x}^{(i)})\right]$
absolute error	$\sum_{i=1}^{n} \left\lvert y^{(i)} - h(\mathbf{x}^{(i)}) \right\rvert$
squared error	$\sum_{i=1}^{n} \left(y^{(i)} - h(\mathbf{x}^{(i)})\right)^2$
worst case	$\max_{1 \leq i \leq n} \left\lvert y^{(i)} - h(\mathbf{x}^{(i)}) \right\rvert$
likelihood	$\Pr(S \mid h) = \prod_{(\mathbf{x},y) \in S} h(\mathbf{x})_y$
log-likelihood	$\log \Pr(S \mid h) = \sum_{(\mathbf{x},y) \in S} \log h(\mathbf{x})_y$

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

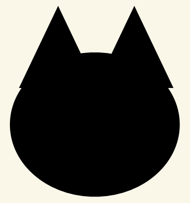
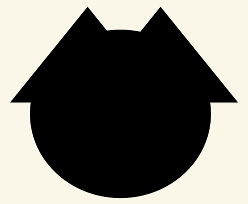
Loss	Definition
0/1 error	$\sum_{i=1}^{n} 1\left[y^{(i)} \neq h(\mathbf{x}^{(i)})\right]$
absolute error	$\sum_{i=1}^{n} \left\lvert y^{(i)} - h(\mathbf{x}^{(i)}) \right\rvert$
squared error	$\sum_{i=1}^{n} \left(y^{(i)} - h(\mathbf{x}^{(i)})\right)^2$
worst case	$\max_{1 \leq i \leq n} \left\lvert y^{(i)} - h(\mathbf{x}^{(i)}) \right\rvert$
likelihood	$\Pr(S \mid h) = \prod_{(\mathbf{x},y) \in S} h(\mathbf{x})_y$
log-likelihood	$\log \Pr(S \mid h) = \sum_{(\mathbf{x},y) \in S} \log h(\mathbf{x})_y$

Probabilistic Predictors

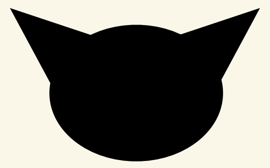
- Rather than predicting **exactly** what a target value will be, many common algorithms predict a **probability distribution** over possible values
    - Especially for **classification** tasks
- Vectors of **indicator variables** are the most common data representation for this scheme:
    - Target features of **training** examples have a single 1 for the **true** value
    - **Predicted** target values are **probabilities** that sum to 1

Probabilistic Predictions Example

Training examples

X	Y_{cat}	Y_{dog}	Y_{panda}
	1	0	0
	0	1	0

Output on test example

X	$h(X)_{cat}$	$h(X)_{dog}$	$h(X)_{panda}$
	0.5	0.45	0.05

Likelihood

For **probabilistic** predictions, we can use **likelihood** to measure the performance of a learning algorithm

Definition:
The **likelihood** for a dataset S of examples and hypothesis h is the **probability** of independently observing the examples according to the probabilities assigned by the **hypothesis**:

$$\Pr(S \mid h) = \prod_{(\mathbf{x},y)\in S} h(\mathbf{x})_y$$

- This has a clear Bayesian interpretation
- We want to maximize likelihood, so it's not a loss (**why?**)
    - **Question:** What is the corresponding loss?
- **Numerical stability issues:** product of probabilities shrinks **exponentially**!
    - *Example:* Probability of **any** sequence of 5000 coin tosses has probability 2^{-5000}!
    - Floating point underflows almost immediately
      (double-precision floating point can't represent anything smaller than 2^{-1021})

Log-Likelihood

Definition:

The log-likelihood for a dataset S of examples and hypothesis h is the log-probability of independently observing the examples according to the probabilities assigned by the hypothesis:

$$\log \Pr(S \mid h) = \log \prod_{(\mathbf{x},y)\in S} h(\mathbf{x})_y$$
$$= \sum_{(\mathbf{x},y)\in S} \log h(\mathbf{x})_y$$

- Taking log of the likelihood fixes the underflow issue (**why**?)
- The log function grows **monotonically**, so maximizing log-likelihood is the **same thing** as maximizing likelihood:

$$\big(\Pr(S \mid h_1) > \Pr(S \mid h_2)\big) \iff \big(\log \Pr(S \mid h_1) > \log \Pr(S \mid h_2)\big)$$

Trivial Predictors

- The simplest possible predictor **ignores all input features** and just predicts the **same value** v for any example
- **Question:** Why would we every want to think about these?

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	$\frac{n_1}{n_0 + n_1}$

Optimal Trivial Predictor Derivations

0/1 error	0 if $n_0 > n_1$ else 1
(negative) log-likelihood	$\frac{n_1}{n_0 + n_1}$

$$L(v) = vn_0 + (1 - v)n_1$$

$$\begin{aligned} L(v) &= -\log \Pr(S \mid v) \\ &= -n_1 \log v - n_0 \log(1 - v) \end{aligned}$$

$$\frac{d}{dv}L(v) = 0$$

$$0 = -\frac{n_1}{v} + \frac{n_0}{1 - v}$$

$$\frac{n_1}{v} = \frac{n_0}{1 - v}$$

$$\frac{n_1}{n_0} = \frac{v}{1 - v} \quad \wedge (0 < v < 1) \implies v = \frac{n_1}{n_0 + n_1}$$

$$\frac{d}{dz}\log z = \frac{1}{z}$$

$$\frac{d}{dz}\log(1 - z) = -\frac{1}{1 - z}$$

Lecture Outline

1. Recap & Logistics
2. Trivial Predictors
3. Linear Regression
4. 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:

$$h(\mathbf{x};\mathbf{w}) = w_0 + w_1 x_1 + \ldots + w_d x_d(e)$$

$$= \sum_{j=0}^{d} w_j x_j$$

For convenience, we often add a special "constant feature" $x_0 = 1$ for all examples

Ordinary Least-Squares

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

1. $L(\mathbf{w}) = \sum_{i=1}^{n} \left(y^{(i)} - h(\mathbf{x}^{(i)}; \mathbf{w}^{(i)})\right)^2 = \sum_{i=1}^{n} \left(y^{(i)} - \sum_{j=0}^{d} w_j^{(i)} x_j^{(i)}\right)^2$

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

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

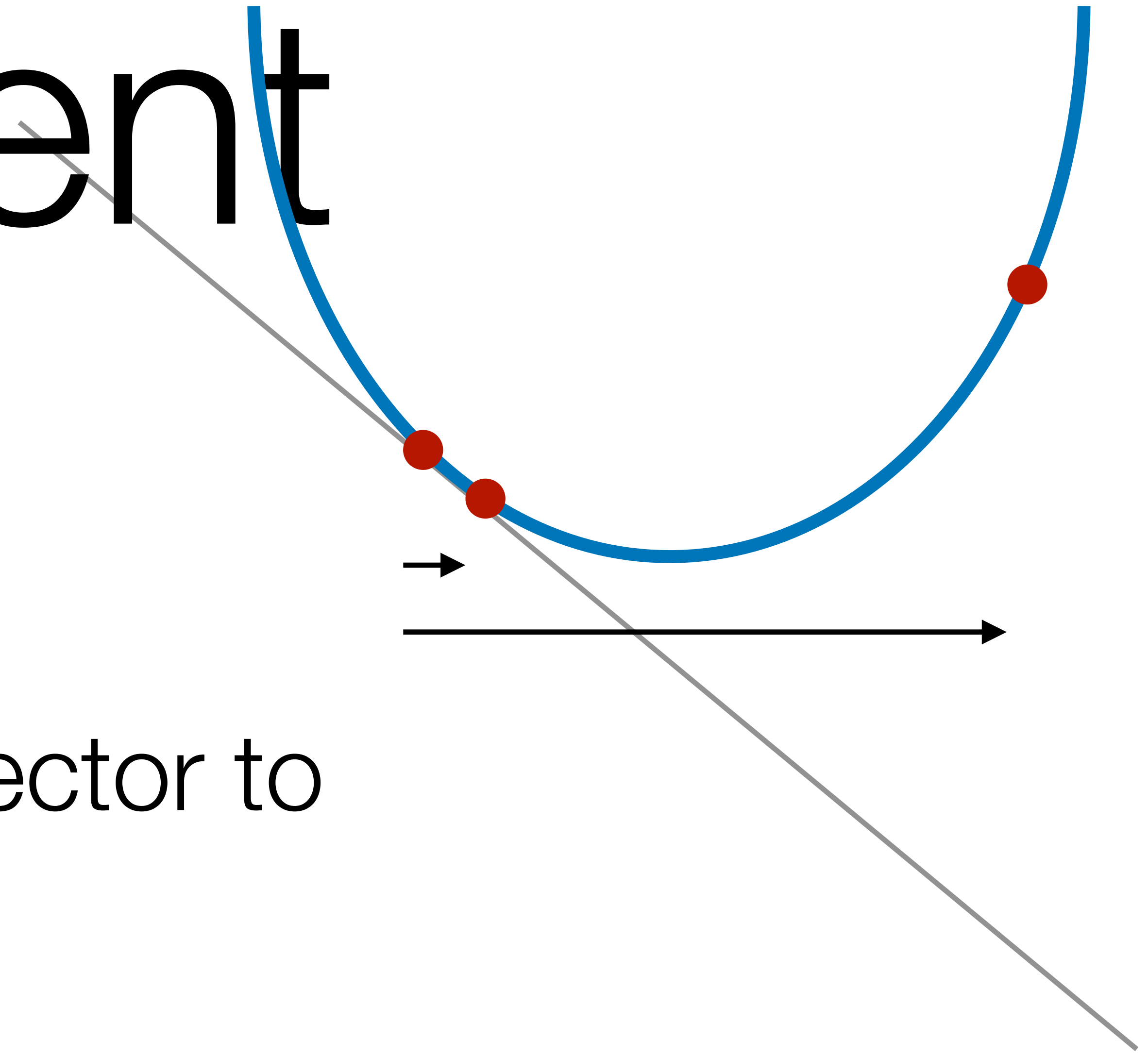
- For d input features, solve a system of $d + 1$ equations
- Requires inverting a $(d+1) \times (d+1)$ matrix $O(d^3)$
- Constructing the matrix requires adding n matrices (one for each example) $O(nd^2)$
- Total cost: $O(nd^2 + d^3)$

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**:

$$w_j^{(t+1)} \leftarrow w_j^{(t)} - \eta \frac{\partial}{\partial w_j^{(t)}} \text{error}(S, \mathbf{w}^{(t)})$$

Recap: 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:

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

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

learning rate

Gradient Descent Variations

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

$$\forall 1 \leq i \leq n : w_j^{(t+1)} \leftarrow w_j^{(t)} - \eta \frac{\partial}{\partial w_j^{(t)}} \text{error}\left(\{(\mathbf{x}^{(i)}, y^{(i)})\}, w^{(t)}\right)$$

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

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

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

$$i \sim U(\{1, \ldots, n\}) : w_j^{(t+1)} \leftarrow w_j^{(t)} - \eta \frac{\partial}{\partial w_j^{(t)}} \text{error}\left(\{(\mathbf{x}^{(i)}, y^{(i)})\}, \mathbf{w}^{(t)}\right)$$

Question

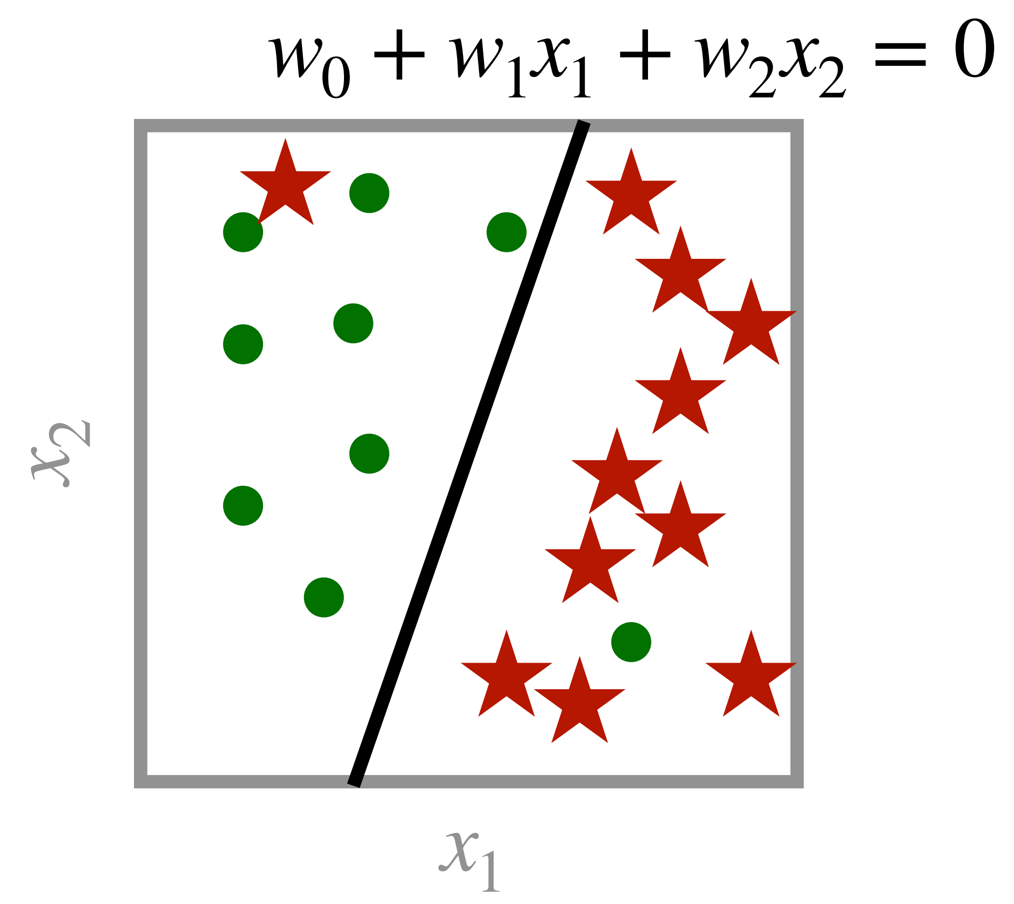
Why would we ever use any of these?

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 $+1$

$$h(\mathbf{x}; \mathbf{w}) = \mathrm{sgn}\left(\sum_{j=0}^{d} w_j x_j\right)$$

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



- The line defined by $\sum_{j=0}^{d} w_j x_j = 0$ 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** $f : \mathbb{R} \rightarrow [0,1]$ instead:

$$h(\mathbf{x}; \mathbf{w}) = f\left(\sum_{j=0}^{d} w_j x_j\right)$$

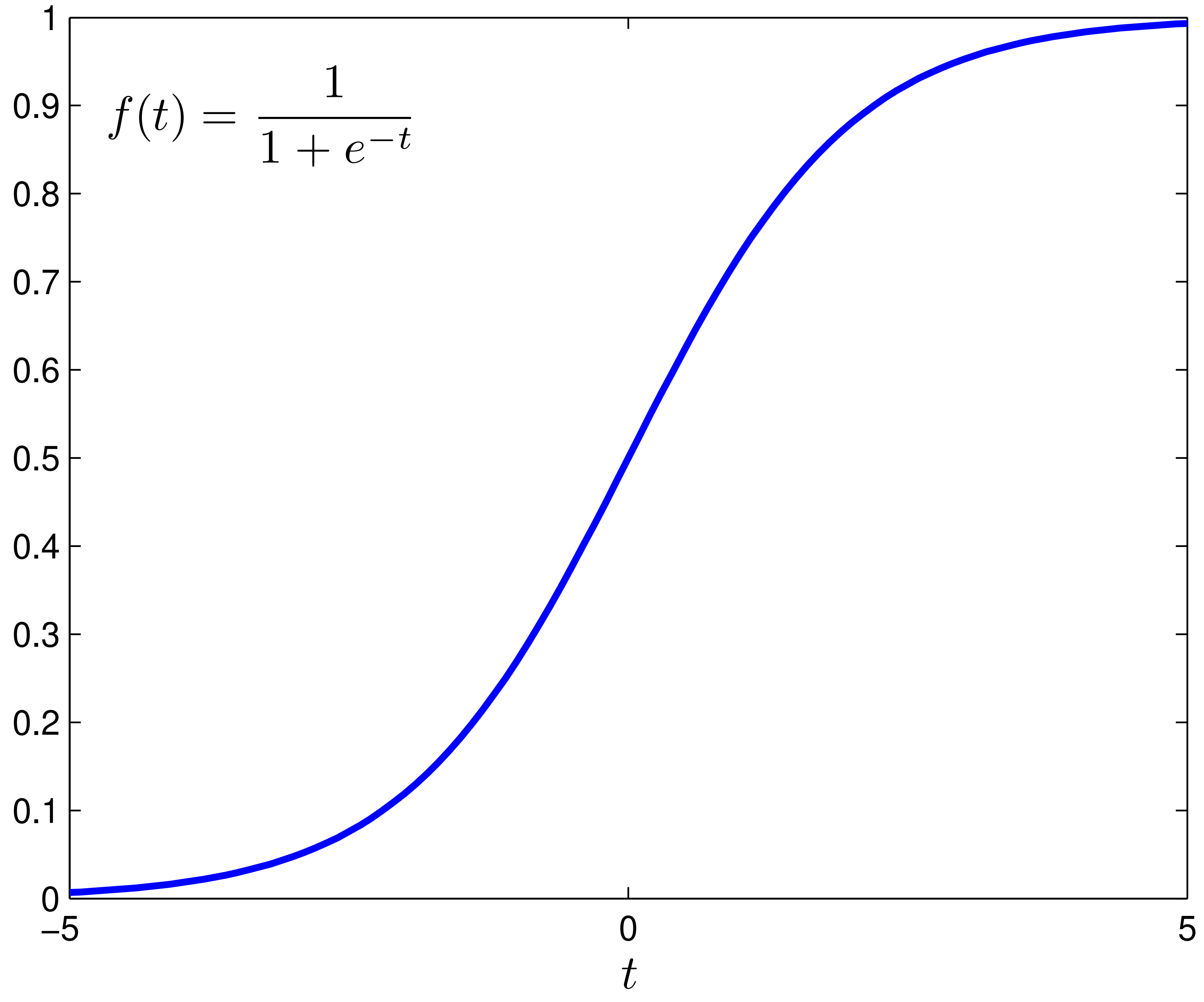
Logistic Regression

- 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:**

$$h(\mathbf{x}; \mathbf{w}) = s\left(\sum_{j=0}^{d} w_j x_j\right)$$



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:

$$h_\ell(\mathbf{x}; \mathbf{w}) = \frac{\exp\left(\sum_{j=0}^{d} w_{\ell,j} x_j\right)}{\sum_{p=1}^{k} \exp\left(\sum_{j=0}^{d} w_{p,j} x_j\right)}$$

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