Linear Models

CMPUT 366: Intelligent Systems

P&M §7.3

Lecture Outline

1. Recap
2. Linear Decision Trees
3. Linear Regression

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 **loss function** for measuring the quality of predictions

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	$\sum_{e \in E} \left\| Y(e) - \hat{Y}(e) \right\| .$
squared error	$\sum_{e \in E} \left(Y(e) - \hat{Y}(e) \right)^2 .$
worst case	$\max_{e \in E} \left\| Y(e) - \hat{Y}(e) \right\| .$
likelihood	$\Pr(E) = \prod_{e \in E} \hat{Y}(e = Y(e))$
log-likelihood	$\log \Pr(E) = \sum_{e \in E} \log \hat{Y}(e = Y(e)) .$

Recap: Optimal Trivial Predictors for Binary Data

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

Loss	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

$$L(v) = vn_1 + (1-v)n_0$$

log-likelihood	$\frac{n_1}{n_0 + n_1}$

$$L(v) = n_1 \log v + n_0 \log(1-v)$$

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

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

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

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

Decision Trees

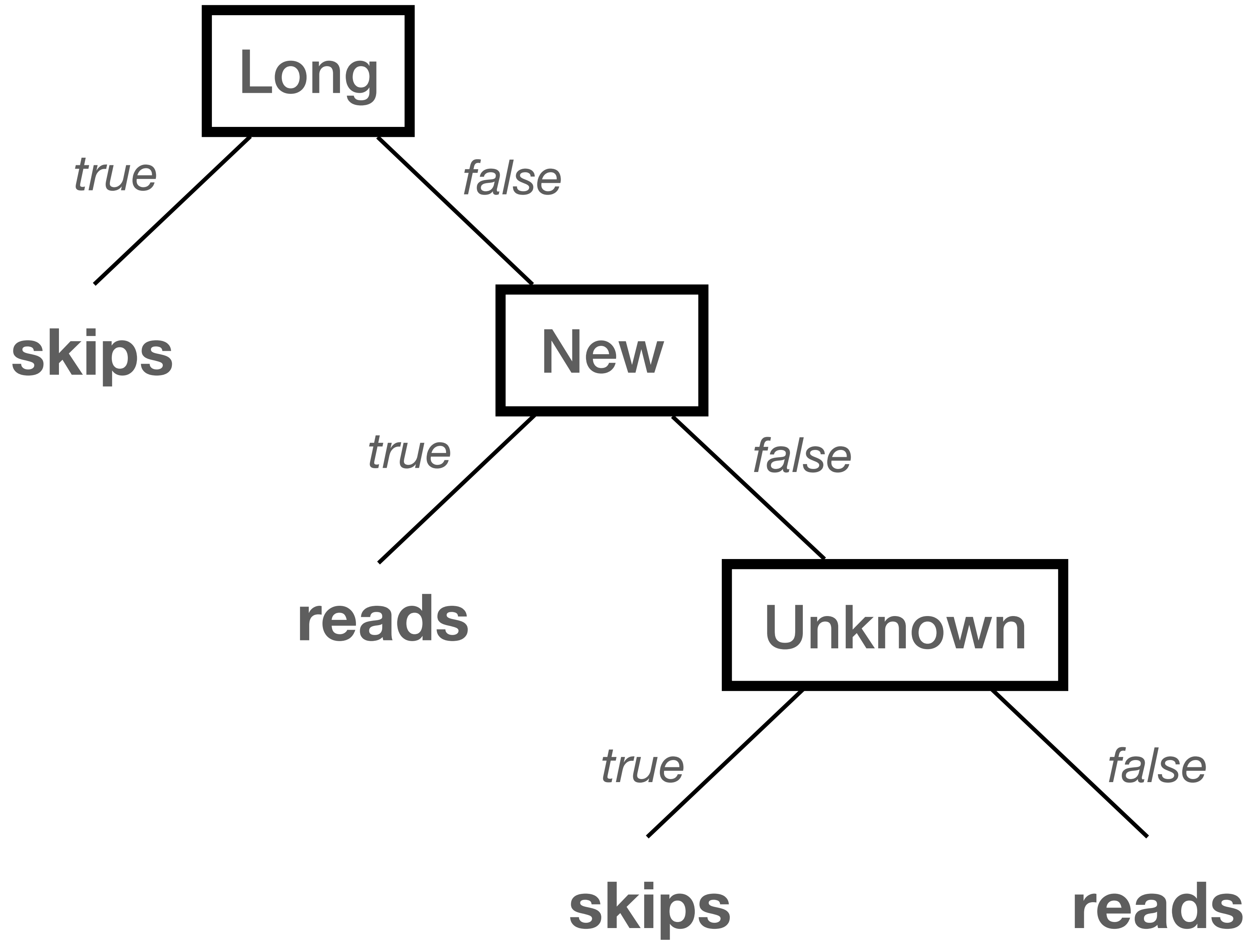
Decision trees are a simple approach to **classification**

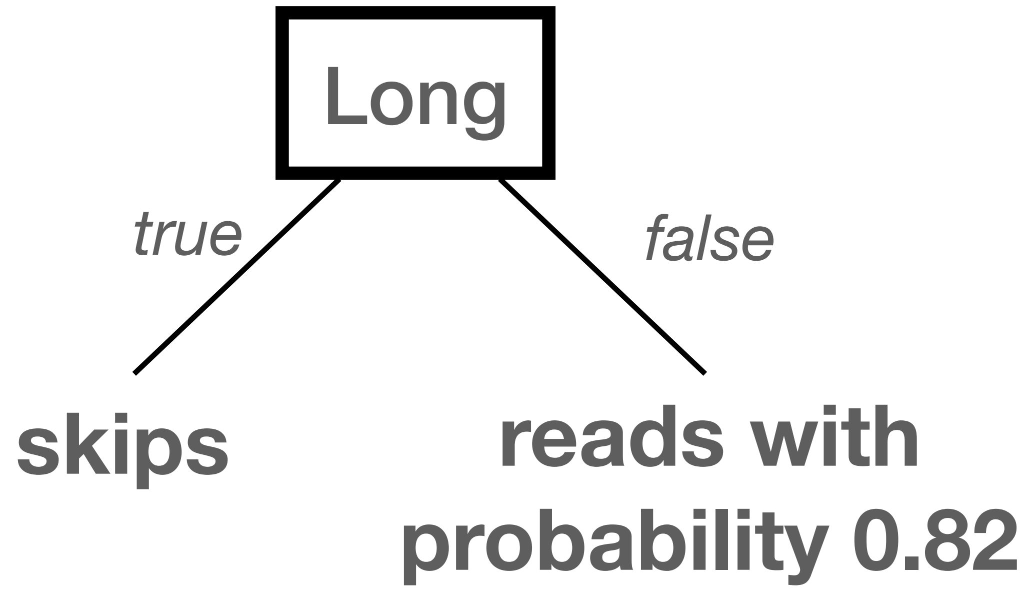
Definition:
A **decision tree** is a tree in which

- Every **internal node** is labelled with a **condition** (Boolean function of an example)
- Every internal node has **two children**, one labelled true and one labelled **false**
- Every leaf node is labelled with a **point estimate** on the **target**

Decision Trees Example

Example	Author	Thread	Length	Where	Action
e1	known	new	long	home	skips
e2	unknown	new	short	work	reads
e3	unknown	followup	long	work	skips
e4	known	followup	long	home	skips
e5	known	new	short	home	reads
e6	known	followup	long	work	skips
e7	unknown	followup	short	work	skips
e8	unknown	new	short	work	reads
e9	known	followup	long	home	skips
e10	known	new	long	work	skips
e11	unknown	followup	short	home	skips
e12	known	new	long	work	skips
e13	known	followup	short	home	reads
e14	known	new	short	work	reads
e15	known	new	short	home	reads
e16	known	followup	short	work	reads
e17	known	new	short	home	reads
e18	unknown	new	short	work	reads





Building Decision Trees

How should an agent **choose** a decision tree?

- **Bias:** which decision trees are preferable to others?
- **Search:** How can we search the space of decision trees?
  - Search space is **prohibitively large**
  - Idea: Choose **features** to branch on **one by one**

Tree Construction Algorithm

learn_tree(*Cs*, *Y*, *Es*):
Input: conditions *Cs*; target feature *Y*; training examples *Es*

if stopping condition is true:
    v := point_estimate(Y, Es)
    $T(e) := v$
    return T
else:
    select condition $c \in Cs$
    $true\_examples := \{ e \in Es \mid c(e) \}$
    t_1 := **learn_tree**($Cs \setminus \{c\}$, Y, $true\_examples$)
    $false\_examples := \{ e \in Es \mid \neg c(e) \}$
    t_0 := **learn_tree**($Cs \setminus \{c\}$, Y, $false\_examples$)
    $T(e)$:= **if** $c(e)$ **then** t_1 **else** t_0
    return T

Tree Construction Algorithm

learn_tree(*Cs*, *Y*, *Es*):
Input: conditions *Cs*; target feature *Y*; training examples *Es*

if stopping condition is true:
    v := point_estimate(*Y*, *Es*)
    T(*e*) := *v*
    return *T*
else:
    select condition $c \in Cs$
    $true\_examples := \{ e \in Es \mid c(e) \}$
    t_1 := **learn_tree**($Cs \setminus \{c\}$, Y, $true\_examples$)
    $false\_examples := \{ e \in Es \mid \neg c(e) \}$
    t_0 := **learn_tree**($Cs \setminus \{c\}$, Y, $false\_examples$)
    T(*e*) := **if** *c*(*e*) **then** t_1 **else** t_0
    return *T*

Unspecified

Stopping Criterion

- **Question:** When **must** the algorithm stop?
    - No more **conditions**
    - No more **examples**
    - All examples have the **same label**
- Additional possible criteria:
    - **Minimum child size:** Do not split a node if there would be too few examples in one of the children (**why**?)
    - **Minimum number of examples:** Do not split a node with too few examples (**why**?)
    - **Improvement criteria:** Do not split a node unless it improves some criterion sufficiently (**why**?)
    - **Maximum depth:** Do not split if the depth reaches a maximum (**why**?)

Leaf Point Estimates

- **Question:** What point estimate should go on the leaves?
    - **Modal** target value
    - **Median** target value (*unless categorical*)
    - **Mean** target value (*unless categorical or ordinal*)
    - **Distribution** over target values
- **Question:** What point estimate **optimally** classifies the leaf's examples?

Split Conditions

- **Question:** What should the set of **conditions** be?
  - **Boolean** features can be used directly
  - **Partition** domain into subsets
    - E.g., **thresholds** for ordered features
  - One **branch for each** domain element

Choosing Split Conditions

- **Question:** Which condition should be chosen to split on?
- Standard answer: **myopically optimal** condition
    - If this was the **only** split, which condition would result in the best performance?

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:

$$\begin{aligned}\hat{Y}^{w}(e) &= w_0 + w_1 X_1(e) + \ldots + w_n X_n(e)\\ &= \sum_{i=0}^{n} w_i X_i(e)\end{aligned}$$

Gradient Descent

- For some loss functions (e.g., sum of squares), linear regression has a **closed-form solution**
- For others, we use **gradient descent**
    - Gradient descent is an iterative method to find the minimum of a function.
    - For **minimizing error**:

$$w_i := w_i - \eta \frac{\partial}{\partial w_i} error(E, w)$$

Gradient Descent Variations

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

$$\forall e_j \in E : w_i := w_i - \eta \frac{\partial}{\partial w_i} error(\{e_j\}, w)$$

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

$$\forall E_j : w_i := w_i - \eta \frac{\partial}{\partial w_i} error(E_j, w)$$

- **Stochastic gradient descent:** repeatedly choose example(s) at random to update on

Linear Classification

- For **binary targets** represented by {0,1} and **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:

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

Logistic Regression

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

$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

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

Linear Regression Trees

- Learning algorithms can be **combined**
- Example: **Linear classification trees**
  - Learn a decision tree until stopping criterion
  - If there are still features left in the leaf, learn a **linear classifier** on the **remaining features**
- Example: **Linear regression trees**
  - Learn a decision tree with **linear regression** in the **leaves**
  - Splitting criterion has to perform linear regression for **each considered split**

Summary

- Decision trees:
  - Split on a **condition** at each internal node
  - Prediction on the **leaves**
  - Simple, general; often a **building block** for other methods
- Linear Regression and Classification
  - Fit a **linear function** to the input and target features
  - Often trained by **gradient descent**
  - For some loss functions, linear regression has a **closed analytic form**