Learning Objectives

At the end of the class you should be able to:

- show an example of decision-tree learning
- explain how to avoid overfitting in decision-tree learning
- explain the relationship between linear and logistic regression
- explain how overfitting can be avoided

Basic Models for Supervised Learning

Many learning algorithms can be seen as deriving from:

- decision trees
- linear (and non-linear) classifiers
- Bayesian classifiers

Learning Decision Trees

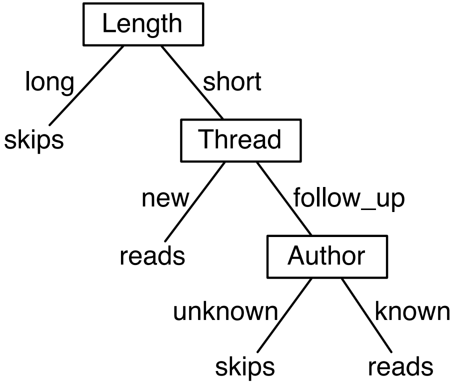
- Representation is a decision tree.
- Bias is towards simple decision trees.
- Search through the space of decision trees, from simple decision trees to more complex ones.

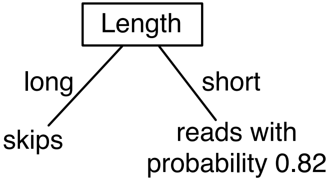
Decision trees

A (binary) decision tree (for a particular output feature) is a tree where:

- Each nonleaf node is labeled with an test (function of input features).
- The arcs out of a node labeled with values for the test.
- The leaves of the tree are labeled with point prediction of the output feature.

Example Decision Trees





Equivalent Logic Program

$$\begin{aligned}
&\mathit{skips} \leftarrow \mathit{long}.\\
&\mathit{reads} \leftarrow \mathit{short} \land \mathit{new}.\\
&\mathit{reads} \leftarrow \mathit{short} \land \mathit{follow\_up} \land \mathit{known}.\\
&\mathit{skips} \leftarrow \mathit{short} \land \mathit{follow\_up} \land \mathit{unknown}.
\end{aligned}$$

or with negation as failure:

$$\begin{aligned}
&\mathit{reads} \leftarrow \mathit{short} \land \mathit{new}.\\
&\mathit{reads} \leftarrow \mathit{short} \land {\sim}\mathit{new} \land \mathit{known}.
\end{aligned}$$

Issues in decision-tree learning

- Given some training examples, which decision tree should be generated?
- A decision tree can represent any discrete function of the input features.
- You need a bias. Example, prefer the smallest tree. Least depth? Fewest nodes? Which trees are the best predictors of unseen data?
- How should you go about building a decision tree? The space of decision trees is too big for systematic search for the smallest decision tree.

Searching for a Good Decision Tree

- The input is a set of input features, a target feature and, a set of training examples.
- Either:
  - Stop and return the a value for the target feature or a distribution over target feature values
  - Choose a test (e.g. an input feature) to split on. For each value of the test, build a subtree for those examples with this value for the test.

Choices in implementing the algorithm

- When to stop:

Choices in implementing the algorithm

- When to stop:
  - no more input features
  - all examples are classified the same
  - too few examples to make an informative split

Choices in implementing the algorithm

- When to stop:
  - no more input features
  - all examples are classified the same
  - too few examples to make an informative split
- Which test to split on isn't defined. Often we use myopic split: which single split gives smallest error.
- With multi-valued features, the text can be can to split on all values or split values into half. More complex tests are possible.

Example Classification Data

Training Examples:

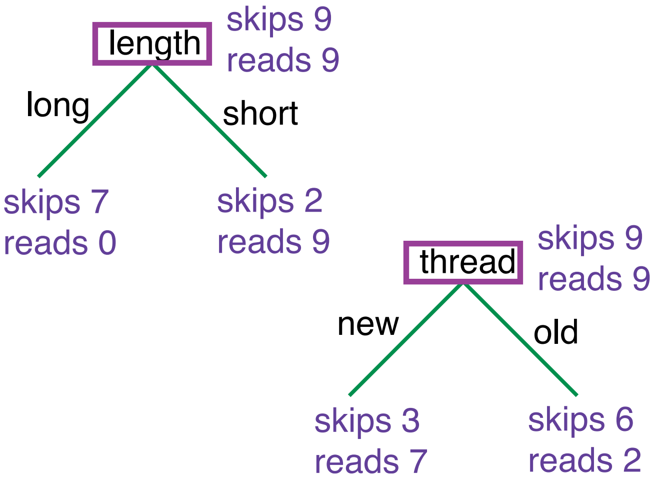
	Action	Author	Thread	Length	Where
e1	skips	known	new	long	home
e2	reads	unknown	new	short	work
e3	skips	unknown	old	long	work
e4	skips	known	old	long	home
e5	reads	known	new	short	home
e6	skips	known	old	long	work

New Examples:

e7	???	known	new	short	work
e8	???	unknown	new	short	work

We want to classify new examples on feature *Action* based on the examples' *Author*, *Thread*, *Length*, and *Where*.

Example: possible splits



Handling Overfitting

- This algorithm can overfit the data.
  This occurs when

Handling Overfitting

- This algorithm can overfit the data.
  This occurs when noise and correlations in the training set that are not reflected in the data as a whole.
- To handle overfitting:
  - restrict the splitting, and split only when the split is useful.
  - allow unrestricted splitting and prune the resulting tree where it makes unwarranted distinctions.
  - learn multiple trees and average them.

Linear Function

A linear function of features $X_1, \ldots, X_n$ is a function of the form:

$$f^{\overline{w}}(X_1, \ldots, X_n) = w_0 + w_1 X_1 + \cdots + w_n X_n$$

We invent a new feature X_0 which has value 1, to make it not a special case.

$$f^{\overline{w}}(X_1, \ldots, X_n) = \sum_{i=0}^{n} w_i X_i$$

Linear Regression

- Aim: predict feature Y from features $X_1, \ldots, X_n$.
- A feature is a function of an example.
  $X_i(e)$ is the value of feature X_i on example e.
- Linear regression: predict a linear function of the input features.

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

$\widehat{Y}^{\overline{w}}(e)$ is the predicted value for Y on example e.
It depends on the weights $\overline{w}$.

Sum of squares error for linear regression

The sum of squares error on examples E for output Y is:

$$
\begin{aligned}
Error_E(\overline{w}) &= \sum_{e \in E} (Y(e) - \widehat{Y}^{\overline{w}}(e))^2 \\
&= \sum_{e \in E} \left(Y(e) - \sum_{i=0}^{n} w_i X_i(e) \right)^2 .
\end{aligned}
$$

Goal: find weights that minimize $Error_E(\overline{w})$.

Finding weights that minimize $Error_E(\overline{w})$

- Find the minimum analytically.
  Effective when it can be done (e.g., for linear regression).

Finding weights that minimize $Error_E(\overline{w})$

- Find the minimum analytically.
  Effective when it can be done (e.g., for linear regression).
- Find the minimum iteratively.
  Works for larger classes of problems.
  Gradient descent:

$$w_i \leftarrow w_i - \eta \frac{\partial Error_E(\overline{w})}{\partial w_i}$$

η is the gradient descent step size, the learning rate.

Linear Classifier

- Assume we are doing binary classification, with classes $\{0, 1\}$ (e.g., using indicator functions).

Linear Classifier

- Assume we are doing binary classification, with classes $\{0, 1\}$ (e.g., using indicator functions).
- There is no point in making a prediction of less than 0 or greater than 1.
- A squashed linear function is of the form:

$$f^{\overline{w}}(X_1, \dots, X_n) = f(w_0 + w_1 X_1 + \cdots + w_n X_n)$$

where f is an activation function.

Linear Classifier

- Assume we are doing binary classification, with classes $\{0,1\}$ (e.g., using indicator functions).
- There is no point in making a prediction of less than 0 or greater than 1.
- A squashed linear function is of the form:

$$f^{\overline{w}}(X_1,\ldots,X_n) = f(w_0 + w_1X_1 + \cdots + w_nX_n)$$

  where f is an activation function.
- A simple activation function is the step function:

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

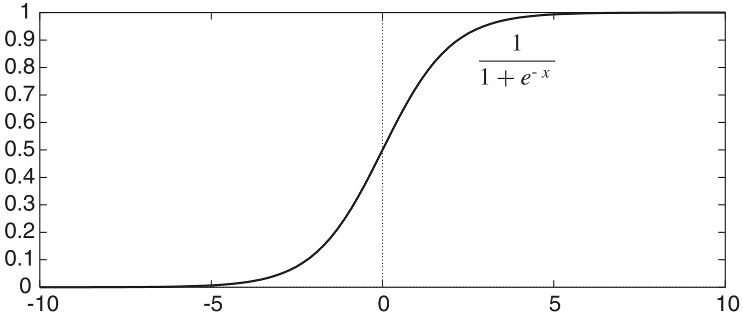
Error for Squashed Linear Function

The sum of squares error is:

$$Error_E(\overline{w}) = \sum_{e \in E} \left(Y(e) - f(\sum_i w_i X_i(e)) \right)^2 .$$

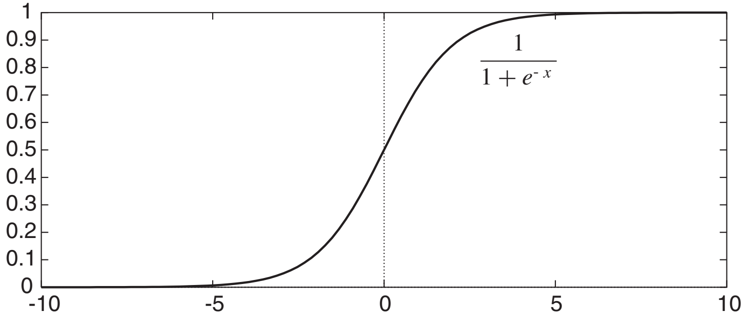
If f is differentiable, we can do gradient descent.

The sigmoid or logistic activation function



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

The sigmoid or logistic activation function



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

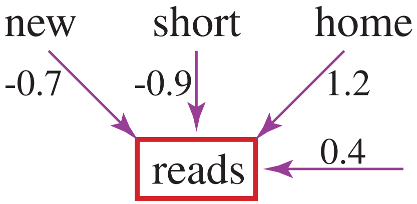
$$f'(x) = f(x)(1 - f(x))$$

A logistic function is the sigmoid of a linear function.
Logistic regression: find weights to minimise error of a logistic function.

Gradient Descent for Logistic Regression

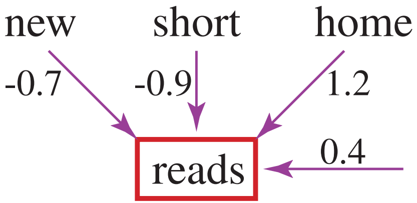
```
1: procedure $LogisticRegression(X, Y, E, \eta)$
2:     • $X$: set of input features, $X = \{X_1, \ldots, X_n\}$
3:     • $Y$: output feature
4:     • $E$: set of examples
5:     • $\eta$: learning rate
6:     initialize $w_0, \ldots, w_n$ randomly
7:     repeat
8:         for each example $e$ in $E$ do
9:             $p \leftarrow f(\sum_i w_i X_i(e))$
10:            $\delta \leftarrow Y(e) - p$
11:            for each $i \in [0, n]$ do
12:                $w_i \leftarrow w_i + \eta\delta p(1-p) X_i(e)$
13:    until some stopping criterion is true
14:    return $w_0, \ldots, w_n$
```

Simple Example



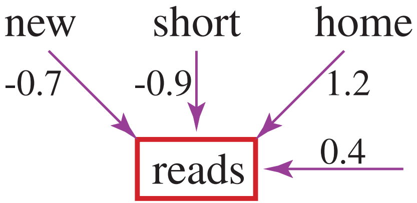
Ex	new	short	home	reads		δ	error
				Predicted	Obs		
e1	0	0	0	$f(0.4) = 0.6$	0	-0.6	0.36
e2	1	1	0		0		
e3	1	0	1		1		

Simple Example



Ex	new	short	home	reads		δ	error
				Predicted	Obs		
e1	0	0	0	$f(0.4) = 0.6$	0	-0.6	0.36
e2	1	1	0	$f(-1.2) = 0.23$	0		
e3	1	0	1	$f(0.9) = 0.71$	1		

Simple Example



Ex	new	short	home	reads		δ	error
				Predicted	Obs		
e1	0	0	0	$f(0.4) = 0.6$	0	-0.6	0.36
e2	1	1	0	$f(-1.2) = 0.23$	0	-0.23	0.053
e3	1	0	1	$f(0.9) = 0.71$	1	0.29	0.084

Linearly Separable

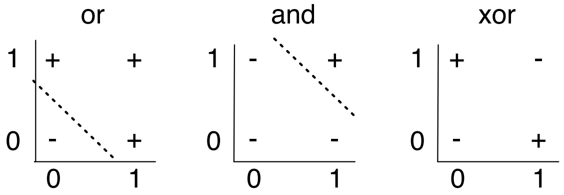
- A classification is linearly separable if there is a hyperplane where the classification is *true* on one side of the hyperplane and *false* on the other side.
- For the sigmoid function, the hyperplane is when:

Linearly Separable

- A classification is linearly separable if there is a hyperplane where the classification is *true* on one side of the hyperplane and *false* on the other side.
- For the sigmoid function, the hyperplane is when:

$$w_0 + w_1X_1 + \cdots + w_nX_n = 0$$

  This separates the predictions > 0.5 and < 0.5.
- linearly separable implies the error can be arbitrarily small

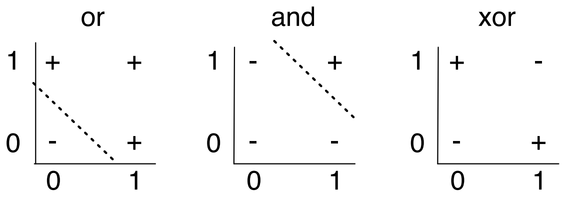


Linearly Separable

- A classification is linearly separable if there is a hyperplane where the classification is *true* on one side of the hyperplane and *false* on the other side.
- For the sigmoid function, the hyperplane is when:

$$w_0 + w_1X_1 + \cdots + w_nX_n = 0$$

This separates the predictions > 0.5 and < 0.5.
- linearly separable implies the error can be arbitrarily small



Kernel Trick: use functions of input features (e.g., product)

Variants in Linear Separators

Which linear separator to use can result in various algorithms:

- Perceptron
- Logistic Regression
- Support Vector Machines (SVMs)
- ...

Bias in linear classifiers and decision trees

- It's easy for a logistic function to represent "at least two of $X_1, \ldots, X_k$ are true":

w_0	w_1	$\cdots$	w_k

Bias in linear classifiers and decision trees

- It's easy for a logistic function to represent "at least two of $X_1, \ldots, X_k$ are true":

w_0	w_1	$\cdots$	w_k
-15	10	$\cdots$	10

  This concept forms a large decision tree.
- Consider representing a conditional: "If X_7 then X_2 else X_3":
  - Simple in a decision tree.
  - Complicated (possible?) for a linear separator

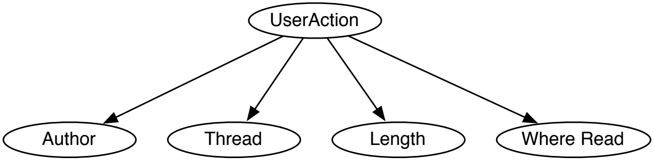
Bayesian classifiers

- Idea: if you knew the classification you could predict the values of features.

$$P(Class|X_1 \dots X_n) \propto P(X_1, \dots, X_n|Class)P(Class)$$

- Naive Bayesian classifier: X_i are independent of each other given the class.
  Requires: $P(Class)$ and $P(X_i|Class)$ for each X_i.

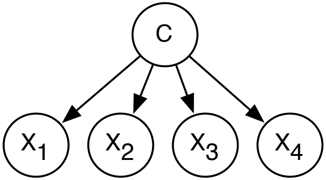
$$P(Class|X_1 \dots X_n) \propto \prod_i P(X_i|Class)P(Class)$$



Learning Probabilities

X_1	X_2	X_3	X_4	C	$Count$
⋮	⋮	⋮	⋮	⋮	⋮
t	f	t	t	1	40
t	f	t	t	2	10
t	f	t	t	3	50
⋮	⋮	⋮	⋮	⋮	⋮

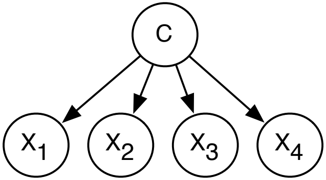
$\longrightarrow$



Learning Probabilities

X_1	X_2	X_3	X_4	C	$Count$
⋮	⋮	⋮	⋮	⋮	⋮
t	f	t	t	1	40
t	f	t	t	2	10
t	f	t	t	3	50
⋮	⋮	⋮	⋮	⋮	⋮

$\longrightarrow$

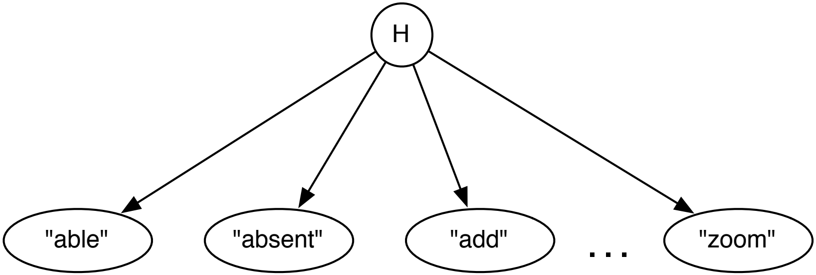


$$P(C{=}v_i) = \frac{\sum_{t \models C=v_i} Count(t)}{\sum_t Count(t)}$$

$$P(X_k = v_j | C{=}v_i) = \frac{\sum_{t \models C=v_i \wedge X_k=v_j} Count(t)}{\sum_{t \models C=v_i} Count(t)}$$

...perhaps including pseudo-counts

Help System



- The domain of H is the set of all help pages.
  The observations are the words in the query.
- What probabilities are needed?
  What pseudo-counts and counts are used?
  What data can be used to learn from?