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

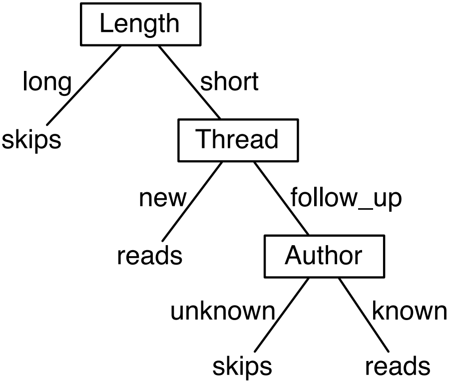
- Representation is a decision tree.
- Bias is towards simple decision trees.
- Search through the space of decision trees, from simple decision trees to more complex ones.

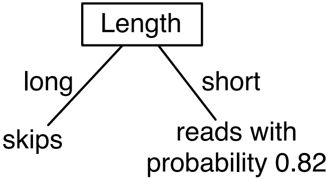
Decision trees

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

- Each nonleaf node is labeled with an input feature.
- The arcs out of a node labeled with feature A are labeled with each possible value of the feature A.
- The leaves of the tree are labeled with point prediction of the output feature.

Example Decision Trees





Equivalent Logic Program

$$skips \leftarrow long.$$
$$reads \leftarrow short \wedge new.$$
$$reads \leftarrow short \wedge follow\_up \wedge known.$$
$$skips \leftarrow short \wedge follow\_up \wedge unknown.$$

or with negation as failure:

$$reads \leftarrow short \wedge new.$$
$$reads \leftarrow short \wedge \sim new \wedge known.$$

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 an input feature to split on.
    For each value of this feature, build a subtree for those examples with this value for the input feature.

Choices in implementing the algorithm

- When to stop:

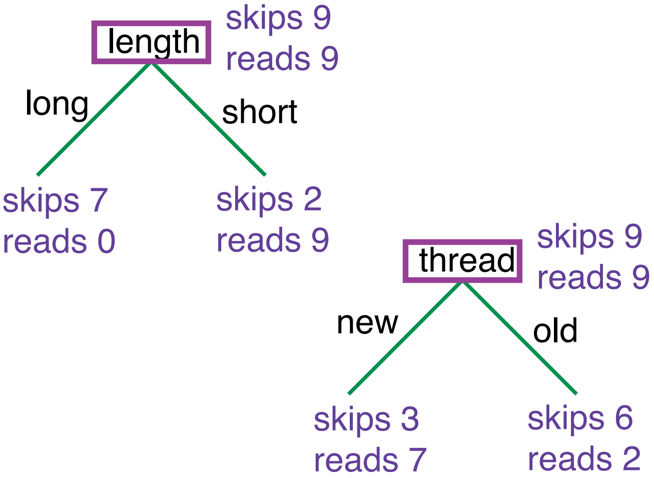
Choices in implementing the algorithm

- 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 feature to select to split on isn't defined. Often we use myopic split: which single split gives smallest error.
- With multi-valued features, we can split on all values or split values into half.

Example: possible splits



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 $X_1, \dots, X_n$ is a function of the form:

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

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

Linear Regression

Linear regression is where the output is a linear function of the input features.

$$pval^{\overline{w}}(e, Y) = w_0 + w_1 val(e, X_1) + \cdots + w_n val(e, X_n)$$

Linear Regression

Linear regression is where the output is a linear function of the input features.

$$pval^{\overline{w}}(e, Y) = w_0 + w_1 val(e, X_1) + \cdots + w_n val(e, X_n)$$

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

$$Error_E(\overline{w}) = \sum_{e \in E} (val(e, Y) - pval^{\overline{w}}(e, Y))^2$$

$$= \sum_{e \in E} (val(e, Y) - (w_0 + w_1 val(e, X_1) + \cdots + w_n val(e, X_n)))^2$$

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.

Gradient Descent for Linear Regression

```
procedure LinearLearner(X, Y, E, η)
    Inputs
        X: set of input features, X = {X_1, ..., X_n}
        Y: output feature
        E: set of examples from which to learn
        η: learning rate
    initialize w_0, ..., w_n randomly
    repeat
        for each example e in E do
            δ ← val(e, Y) − pval^w̄(e, Y)
            for each i ∈ [0, n] do
                w_i ← w_i + η δ val(e, X_i)
    until some stopping criterion is true
    return w_0, ..., w_n
```

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:

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

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

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

Gradient Descent for Linear Classifiers

If the activation is differentiable, we can use gradient descent to update the weights. The sum of squares error is:

$$Error_E(\overline{w}) = \sum_{e \in E} (val(e, Y) - f(\sum_i w_i \times val(e, X_i)))^2$$

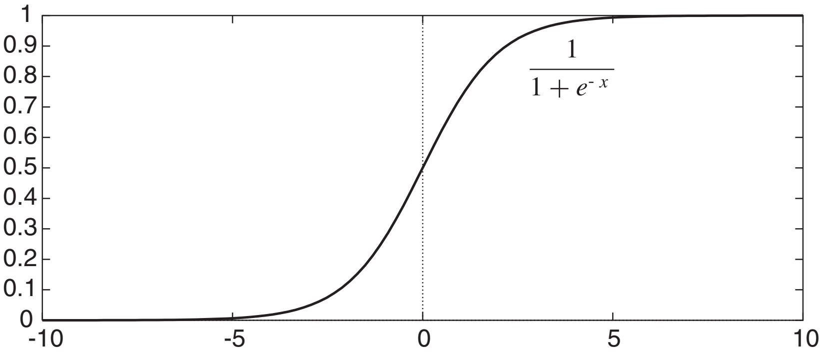
The partial derivative with respect to weight w_i is:

$$\frac{\partial Error_E(\overline{w})}{\partial w_i} = -2 \times \delta \times f'(\sum_i w_i \times val(e, X_i)) \times val(e, X_i)$$

where $\delta = val(e, Y) - pval^{\overline{w}}(e, Y)$.
Thus, each example e updates each weight w_i by

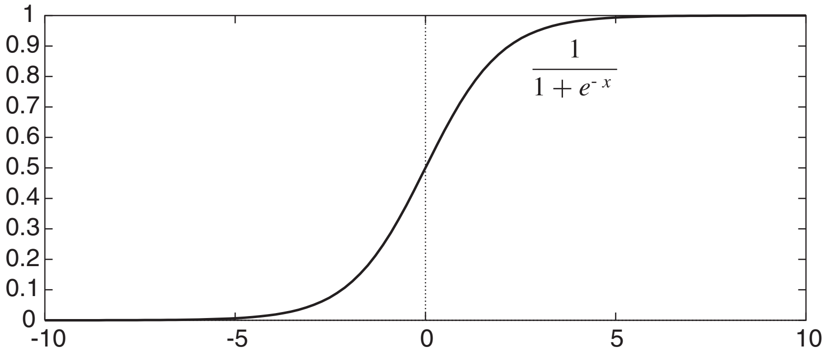
$$w_i \leftarrow w_i + \eta \times \delta \times f'(\sum_i w_i \times val(e, X_i)) \times val(e, X_i)$$

The sigmoid or logistic activation function



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

The sigmoid or logistic activation function



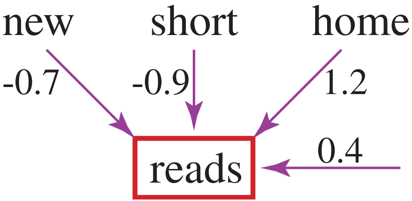
$$f(x) = \frac{1}{1 + e^{-x}}$$

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

Gradient Descent for Logistic Regression

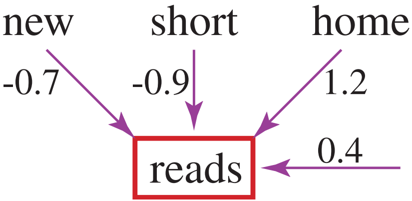
procedure $LinearLearner(X, Y, E, \eta)$
    Inputs
        X: set of input features, $X = \{X_1, \ldots, X_n\}$
        Y: output feature
        E: set of examples from which to learn
        η: learning rate
    initialize $w_0, \ldots, w_n$ randomly
    repeat
        for each example e in E **do**
            $p \leftarrow f(\sum_i w_i \times val(e, X_i))$
            $\delta \leftarrow val(e, Y) - p$
            for each $i \in [0, n]$ **do**
                $w_i \leftarrow w_i + \eta\delta p(1 - p) val(e, X_i)$
    until some stopping criterion is true
    return $w_0, \ldots, w_n$

Simple Example



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

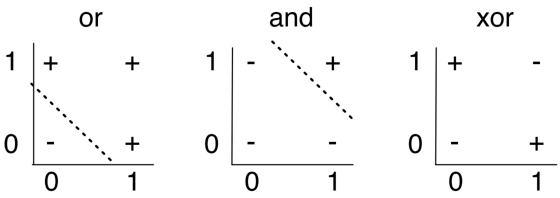
Simple Example



Ex	new	short	home	reads		error
				Predicted	Obs	
e1	0	0	0	$f(0.4) = 0.6$	0	0.36
e2	1	1	0	$f(-1.2) = 0.23$	0	0.053
e3	1	0	1	$f(0.9) = 0.71$	1	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: $w_0 + w_1 \times val(e, X_1) + \cdots + w_n \times val(e, X_n) = 0$.
- If the data are linearly separable, the error can be made arbitrarily small.



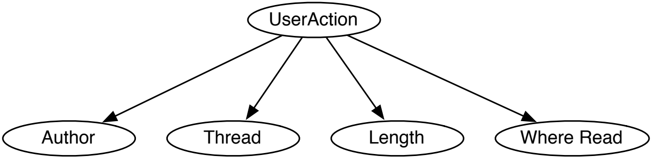
Bayesian classifiers

- 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.

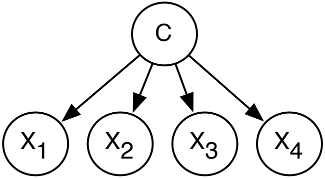
$$P(Class|X_1 \dots X_n) \propto \prod_i P(X_i|Class)P(Class)$$



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
⋮	⋮	⋮	⋮	⋮	⋮

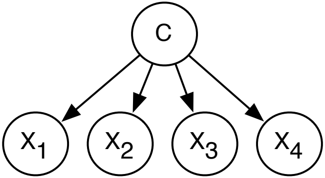
$\longrightarrow$



Learning Probabilities

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

$\longrightarrow$

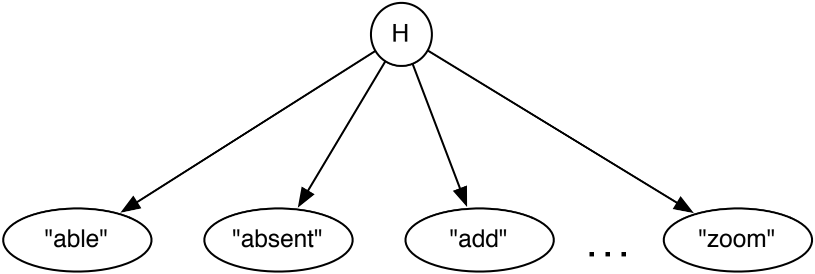


$$P(C{=}v_i) = \frac{\sum_{t \models C=v_i} Count(t)}{\sum_t Count(t)}$$

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

...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?