Basic Models for Supervised Learning

Many learning algorithms can be seen as deriving from:

- decision trees
- linear classifiers
- Bayesian classifiers

Learning Decision Trees

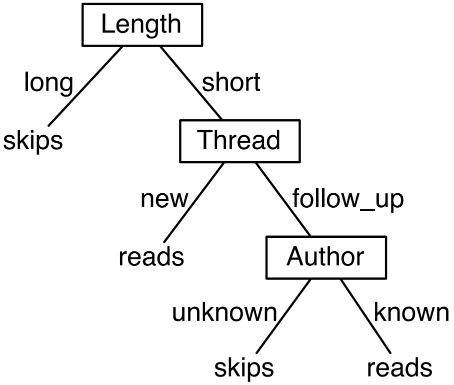
- Representation is a decision tree.
- Bias is towards simple decision trees.
- Search through the space of decision trees, from simple decision trees to more complex ones.

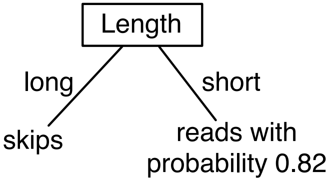
Decision trees

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

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

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.

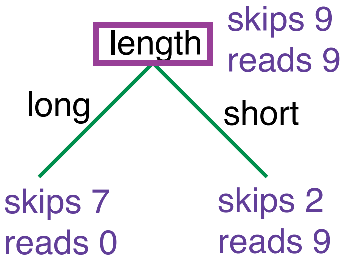
Searching for a Good Decision Tree

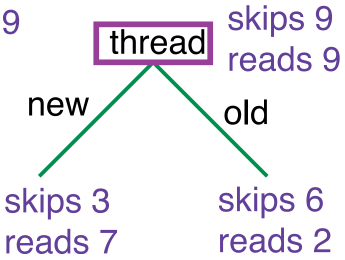
- The input is a set of input features, a target feature and, a set of training examples.
- Stop if all examples have the same classification.
- Otherwise, choose an input feature to split on,
  - for each value of this feature, build a subtree for those examples with this value for the input feature.

Using this algorithm in practice

- This assumes input features are adequate to represent the concept. It can can stop earlier and return probabilities at leaves.
- Which feature to select to split on isn't defined. Often we use myopic split: which single split gives smallest error.
- Overfitting is a problem.

Example: possible splits





Handling Overfitting

- This algorithm gets into trouble overfitting the data. This occurs with noise and correlations in the training set that are not reflected in the data as a whole.
- To handle overfitting:
    - You can restrict the splitting, so that you split only when the split is useful.
    - You can allow unrestricted splitting and prune the resulting tree where it makes unwarranted distinctions.

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.

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

```
1:  procedure LinearLearner($X, Y, E, \eta$)
2:          $X$: set of input features, $X = \{X_1, \ldots, X_n\}$
3:          $Y$: output feature
4:          $E$: set of examples from which to learn
5:          $\eta$: learning rate
6:      initialize $w_0, \ldots, w_n$ randomly
7:      repeat
8:          for each example $e$ in $E$ do
9:              $\delta \leftarrow val(e, Y) - pval^{\overline{w}}(e, Y)$
10:             for each $i \in [0, n]$ do
11:                 $w_i \leftarrow w_i + \eta \delta val(e, X_i)$
12:             end for each
13:         end for each
14:     until some stopping criteria is true
15:     return $w_0, \ldots, w_n$
```

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:

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

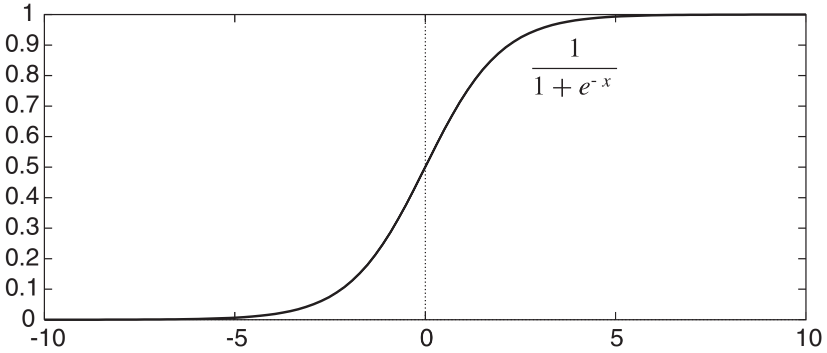
The partial derivative with respect to weight w_i is:

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

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

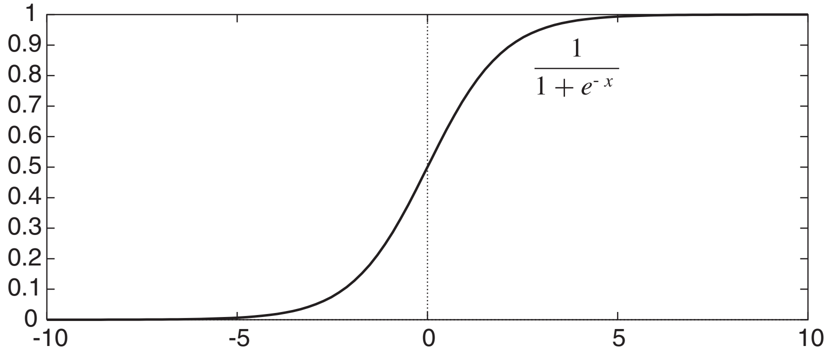
$$w_i \leftarrow w_i + \eta\delta f'(\sum_i w_i \times val(e, X_i)) \times val(e, X_i)$$

The sigmoid or logistic activation function



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

The sigmoid or logistic activation function

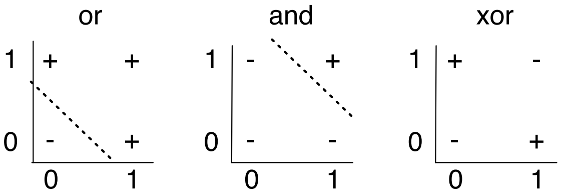


$$f(x) = \frac{1}{1+e^{-x}}$$

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

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.
- The hyperplane is defined by where the predicted value, $f^{\overline{w}}(X_1, \ldots, X_n) = f(w_0 + w_1 val(e, X_1) + \cdots + w_n val(e, X_n))$ is 0.5. For the sigmoid function, the hyperplane is defined by $w_0 + w_1 val(e, X_1) + \cdots + w_n val(e, X_n) = 0$.
- If the data are linearly separable, the error can be made arbitrarily small.



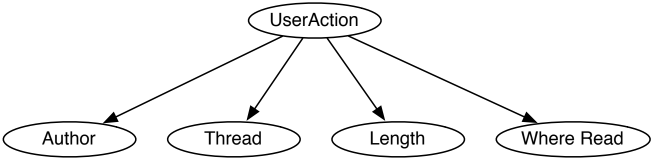
Bayesian classifiers

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

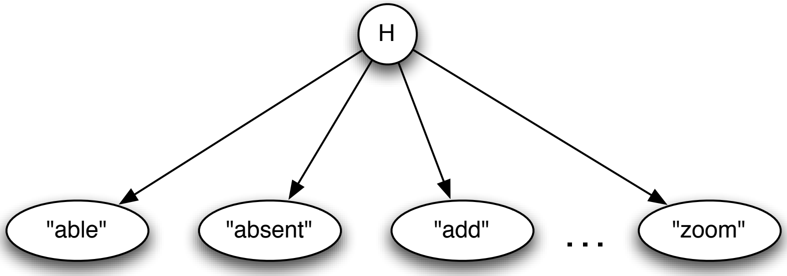
$$P(Class|X_1 \dots X_n) \propto P(X_1, \dots, X_n|Class)P(Class)$$

- 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.

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



Help System


H
"able"
"absent"
"add"
. . .
"zoom"