Learning Objectives

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

- show how decision-tree learning works on small examples
- explain the relationship between linear and logistic regression
- explain the updates of stochastic gradient descent

Basic Models for Supervised Learning

Many learning algorithms can be seen as deriving from:

- decision trees
- linear (and non-linear) 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 target feature) is a tree where:

- each internal (non-leaf) node is labeled with a condition, a Boolean function of examples, built using input features
- each internal node has two branches, one labeled *true* and the other *false*
- each leaf of the tree is labeled with a point estimate of the target feature.

Decision trees are also called classification trees when the target is discrete, and regression trees when the target is real-valued.

Decision trees

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

- each internal (non-leaf) node is labeled with a condition, a Boolean function of examples, built using input features
- each internal node has two branches, one labeled *true* and the other *false*
- each leaf of the tree is labeled with a point estimate of the target feature.

Decision trees are also called classification trees when the target is discrete, and regression trees when the target is real-valued.

- Like an if–then–else structure in a programming language.

Example Classification Data

Training Examples:

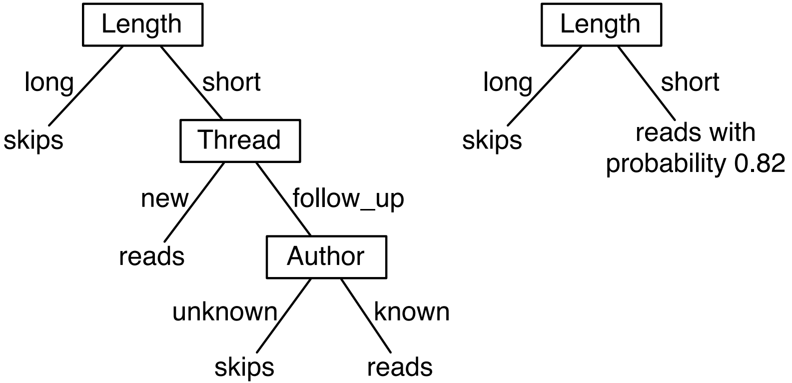
	Action	Author	Thread	Length	Where
e1	skips	known	new	long	home
e2	reads	unknown	new	short	work
e3	skips	unknown	old	long	work
e4	skips	known	old	long	home
e5	reads	known	new	short	home
e6	skips	known	old	long	work

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 Decision Trees



Equivalent Programs

```
define action(e):
    if long(e): return skips
    else if new(e): return reads
    else if unknown(e): return skips
    else: return reads
```

Equivalent Programs

define action(e):
    if $long(e)$: return $skips$
    else if $new(e)$: return $reads$
    else if $unknown(e)$: return $skips$
    else: return $reads$

Logic Program:

$$reads(E) \leftarrow short(E) \wedge new(E).$$

$$reads(E) \leftarrow short(E) \wedge follow\_up(E) \wedge known(E).$$

$$skips(E) \leftarrow long(E).$$

$$skips(E) \leftarrow short(E) \wedge follow\_up(E) \wedge unknown(E).$$

Equivalent Programs

define action(e):
  if $long(e)$: return $skips$
  else if $new(e)$: return $reads$
  else if $unknown(e)$: return $skips$
  else: return $reads$

Logic Program:

$$reads(E) \leftarrow short(E) \land new(E).$$

$$reads(E) \leftarrow short(E) \land follow\_up(E) \land known(E).$$

$$skips(E) \leftarrow long(E).$$

$$skips(E) \leftarrow short(E) \land follow\_up(E) \land unknown(E).$$

or with negation as failure:

$$reads \leftarrow short \land new.$$

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

Equivalent Programs

define action(e):
    if $long(e)$: return *skips*
    else if $new(e)$: return *reads*
    else if $unknown(e)$: return *skips*
    else: return *reads*

Logic Program:

$$reads(E) \leftarrow short(E) \wedge new(E).$$

$$reads(E) \leftarrow short(E) \wedge follow\_up(E) \wedge known(E).$$

$$skips(E) \leftarrow long(E).$$

$$skips(E) \leftarrow short(E) \wedge follow\_up(E) \wedge unknown(E).$$

or with negation as failure:

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

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

or as a logical fromula: $reads \leftrightarrow short \wedge (new \vee known)$

Issues in decision-tree learning

- Given some training examples, which decision tree should be generated?

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.

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.

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?

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?

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 a value for the target feature or a distribution over target feature values
  - Choose a condition (e.g. an input feature) to split on.
    build a subtree for those examples with with the condition true and the examples with the condition false.

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
  - no split give an appreciable improvement in error

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
  - no split give an appreciable improvement in error
- Which test to split on isn't defined. Often we use myopic split: which single split gives smallest error?

Decision_tree_learner

procedure $DT\_learner(Cs, Y, Es, \gamma)$

  Inputs Cs: set of possible conditions; Y: target feature; Es: training examples; γ: improvement threshold

  Output function to predict a value of Y for an example

  $c := select\_split(Es, Cs, \gamma)$ ▷ see next slide

  if $c = None$ **then** ▷ stopping criterion is true

    $v := leaf\_prediction(Y, Es)$ ▷ Prediction on Y

    define $T(e) = v$

    return T

  else

    $true\_examples := \{e \in Es : c(e)\}$

    $t_1 := DT\_learner(Cs \setminus \{c\}, Y, true\_examples, \gamma)$

    $false\_examples := \{e \in Es : \neg c(e)\}$

    $t_0 := DT\_learner(Cs \setminus \{c\}, Y, false\_examples, \gamma)$

    define $T(e) =$ if $c(e)$ then $t_1(e)$ else $t_0(e)$

    return T

```
procedure select_split(Es, Y, Cs, γ)
    best_val := sum_loss(Y, Es) − γ
    best_split := None
    for c ∈ Cs do
        val := sum_loss(Y, {e ∈ Es | c(e)})
                 + sum_loss(Y, {e ∈ Es | ¬c(e)})
        if val < best_val then
            best_val := val
            best_split := c
    return best_split
```

For log loss: Prediction is empirical proportion of Y value

- $P = leaf\_prediction(Y, Es) : v \mapsto \frac{|\{e' \in Es : Y(e) = v\}|}{|Es|}$

$$sum\_loss(Y, Es) = \sum_{e \in Es} log(P(Y(e)))$$

Example Classification Data

Training Examples:

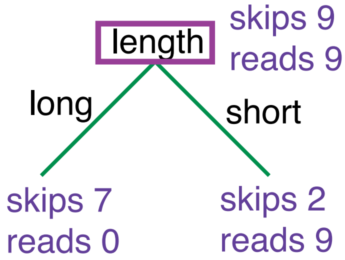
	Action	Author	Thread	Length	Where
e1	skips	known	new	long	home
e2	reads	unknown	new	short	work
e3	skips	unknown	old	long	work
e4	skips	known	old	long	home
e5	reads	known	new	short	home
e6	skips	known	old	long	work

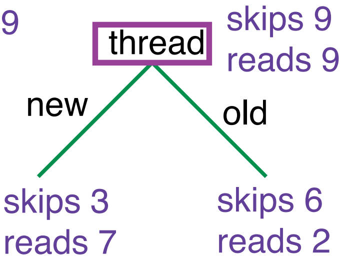
New Examples:

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

Aim: 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 (decision forests, random forests)

Linear Function

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

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

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 target Y is:

$$SSE(E, \overline{w}) = \sum_{e \in E} (Y(e) - \widehat{Y}^{\overline{w}}(e))^2$$

$$= \sum_{e \in E} \left(Y(e) - \sum_{i=0}^{n} w_i * X_i(e) \right)^2 .$$

Goal: given examples E, find weights that minimize $SSE(E, \overline{w})$.

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:

$$w_i \leftarrow w_i - \eta \frac{\partial}{\partial w_i} Error(E, \overline{w})$$

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

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:

$$w_i \leftarrow w_i - \eta \frac{\partial}{\partial w_i} Error(E, \overline{w})$$

  η is the gradient descent step size, the learning rate.
- Often update weights after each example:
  — incremental gradient descent updates parameters after each example
  — stochastic gradient descent updates parameters after a batch of (randomly selected) examples
  Often much faster than updating weights after sweeping through examples, but may not converge to a local optimum

Linear Classifier

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

Linear Classifier

- Assume you 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.

Linear Classifier

- Assume you 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:

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

Cannot be used in gradient descent because

Linear Classifier

- Assume you 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:

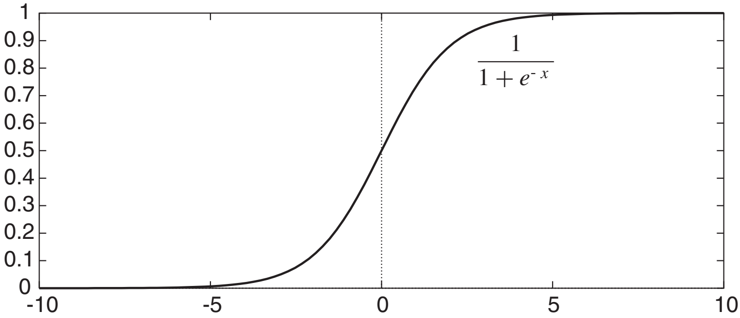
$$f^{\overline{w}}(X_1, \ldots, X_n) = f(w_0 + w_1 X_1 + \cdots + w_n X_n)$$

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

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

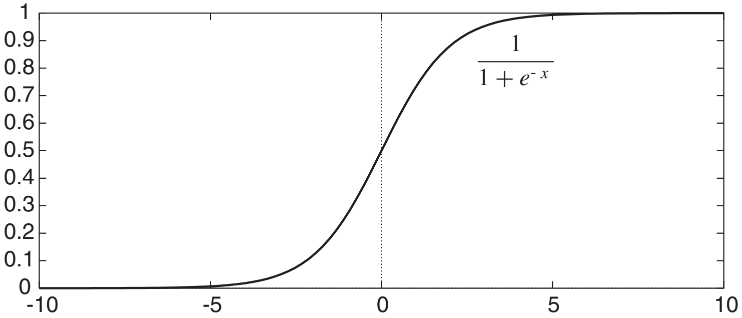
Cannot be used in gradient descent because it has a derivative of 0 almost everywhere (except at 0)

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 minimize log loss of a logistic function.

Error for Squashed Linear Function

When the domain of target Y is $\{0, 1\}$:

- $\widehat{Y}(e) = sigmoid\left(\sum_{i=0}^{n} w_i * X_i(e)\right)$.
- $\delta(e) = Y(e) - \widehat{Y}^{\overline{w}}(e)$

Error for Squashed Linear Function

When the domain of target Y is $\{0,1\}$:

- $\widehat{Y}(e) = sigmoid\left(\sum_{i=0}^{n} w_i * X_i(e)\right)$.
- $\delta(e) = Y(e) - \widehat{Y}^{\overline{w}}(e)$

A natural measure for sigmoid is log loss:

$$LL(E,\overline{w}) = \sum_{e \in E} Y(e) * \log \widehat{Y}(e) + (1 - Y(e)) * \log(1 - \widehat{Y}(e))$$

$$\frac{\partial}{\partial w_i} LL(E,\overline{w}) =$$

Error for Squashed Linear Function

When the domain of target Y is $\{0, 1\}$:

- $\widehat{Y}(e) = sigmoid\left(\sum_{i=0}^{n} w_i * X_i(e)\right)$.
- $\delta(e) = Y(e) - \widehat{Y}^{\overline{w}}(e)$

A natural measure for sigmoid is log loss:

$$LL(E, \overline{w}) = \sum_{e \in E} Y(e) * \log \widehat{Y}(e) + (1 - Y(e)) * \log(1 - \widehat{Y}(e))$$

$$\frac{\partial}{\partial w_i} LL(E, \overline{w}) = \sum_{e \in E} \delta(e) * X_i(e)$$

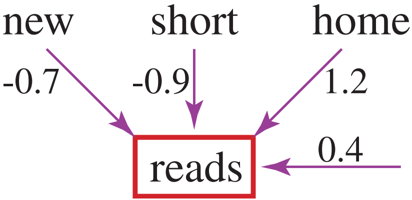
Linear Learner with Stochastic Gradient Descent

procedure *Linear_learner*(Xs, Y, Es, η, b)

- Input features: $Xs = \{X_1, \ldots, X_n\}$. Target feature: Y. Examples: Es. Learning rate: η. Batch size: b

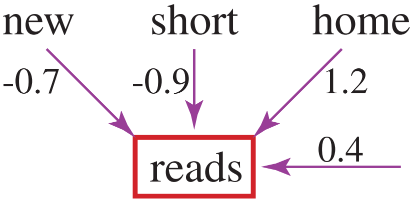
    initialize $w_0, \ldots, w_n$ randomly
    define $pred(e) = \phi(\sum_i w_i * X_i(e))$
    repeat
        for each $i \in [0, n]$ **do** d[i] := 0
        select batch $B \subseteq Es$ of size b
        for each example e in B **do**
            $error := pred(e) - Y(e)$
            for each $i \in [0, n]$ **do**
                $d_i := d_i + error * X_i(e)$
        for each $i \in [0, n]$ **do**
            $w_i := w_i - \eta * d_i / b$
    until termination
    return *pred*

Simple Example



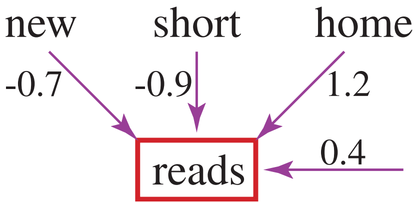
Ex	new	short	home	reads		δ	SSE
				Predicted	Obs		
e1	0	0	0	$f(0.4) \approx 0.6$	0	-0.6	0.36
e2	1	1	0		0		
e3	1	0	1		1		

Simple Example



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

Simple Example



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

Linearly Separable

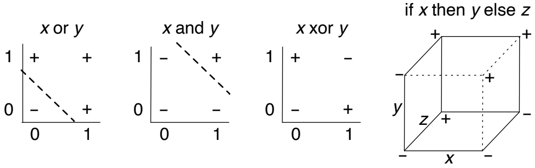
- A classification is linearly separable if there is a hyperplane where the classification is *true* on one side of the hyperplane and *false* on the other side.
- For the sigmoid function, the hyperplane is when:

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:

$$w_0 + w_1 * X_1 + \cdots + w_n * X_n = 0$$

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



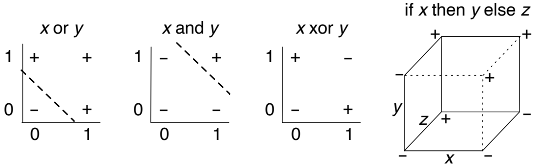
(a) (b) (c) (d)

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:

$$w_0 + w_1 * X_1 + \cdots + w_n * X_n = 0$$

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



(a) (b) (c) (d)

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.
  - For a linear separator

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.
  - For a linear separator it is impossible to represent as it is not linearly separable