• Agents can learn to improve:
  • inference from percepts
  • information about world evolution
    • as the result of a changing world
    • as the result of actions
  • utility estimators
  • action choices
    • either update condition-action maps
    • goal modification to maximize utility
What we want to learn

• Mapping function
  • Inputs are factored representations
e.g. a vector of values
  • Outputs are
    • discrete (e.g. categorical)
    • continuous

Types of learning

• Inductive – Learn map between input/output pairs

• Deductive – Creating rules that are logically entailed, such as if I am in a dark cave and I don’t smell a breeze, I’m not going to step into a pit.
Learners vary based on their feedback

- Unsupervised learning
  - No explicit feedback
  - Goal is to cluster "similar" things

---

Roch et al., unpublished

---

Learners

- Reinforcement learning
  - Learner is given rewards/punishments for actions
  - Example: Positive reinforcement animal training

- Supervised learning
  - Each input is paired with a label or value and the agent attempts to learn to predict the labels/values for novel data.

- Hybrids are possible, such as semi-supervised learning where a small set of labeled data accompanies a large set of unlabeled data.
Caveat about labeled data sets

• We refer to labels as “ground truth.”

• One should be cautious with ground truth...
  Why?

Supervised learning

• Suppose there exists an unknown \( f: x \rightarrow y \) such that
  \[ (y_1 = f(x_1)) \land (y_2 = f(x_2)) \land (y_3 = f(x_3)) \land \ldots \]
  and we are given only a training set
  \[ (x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots \]

• Supervised learning estimates a function \( h: x \rightarrow y \) that approximates \( f \).
Supervised learning

- Function $h$ is the *hypothesis* and our estimation is a search in *hypothesis function space* for a good hypothesis.

- Learning is a search for a good hypothesis.

- How do we measure goodness?
  - Evaluate the function on a labeled test set.
  - The test set must be distinct from the training set: $\text{training} \cap \text{test} = \emptyset$
  - We say $h$ *generalizes* well if it performs well on the test set.

Why do we need to test on novel data?

How to choose amongst functions?

Ockham’s razor – Use simplest hypothesis consistent with the data

All of these functions fit the training data, but which one is most likely to correctly predict new data?
Hypothesis spaces

• The more complex a hypothesis space, the more difficult it is to find a good hypothesis.

• Fits well with Ockham’s Razer.

Supervised learning

• Function \( f \) could be stochastic
  • If so, \( f \) is not simply a function of \( x \)
  • In these cases, we learn a conditional probability distribution \( P(Y|x) \)

• What are we learning:
  • \( y \) is categorical \( \rightarrow \) classification
    • example: \( y \in \{ \text{happy, sad, angry, serious} \} \)
    • binary classifier – special case with exactly two classes
  • \( y \) is numeric \( \rightarrow \) regression
    • example: \( y = \text{change in sea level (m) since 1990} \)
Decision tree learner

• Answers a series of questions to arrive at a solution

• For now, we restrict our discussion to
  • questions that have categorical (discrete) answers
  • binary classification decisions

Dr. Stuart Russell is hungry...

Professor Russell’s decision tree for where to eat...
9 questions from 10 attributes (price is not used)
Learning a tree from examples

Constructing a tree from examples

- Which question to ask first?

- What do you look for when you play 20 questions?

Chances are, you intuitively use information theory...
Quantity of information

• Amount of surprise that one sees when observing an event.

\[ I(x_i) = \log_2 \frac{1}{P(x_i)} \]

• If an event is rare, we can derive a large quantity of information (measured in bits) from it.

Note: We use log base 2 and will start omitting the base later on.

Expectation (review)

• An expected value is the value that we expect to see most often.

• We sum the product of each possible value and the probability that it occurs

\[ E[X] = \sum_{x_j \in S} x_j P(x_j) \text{ where } S \text{ is the set of all possible values of } X \]
Entropy

• Entropy is defined as the expected amount of information (average amount of surprise) and is usually denoted by the symbol $H$.

\[
H(X) = E[I(X)] = \sum_{x_i \in S} P(x_i) I(x_i)
\]

$S$ is all possible symbols

\[
= \sum_{x_i \in S} P(x_i) \log_2 \frac{1}{P(x_i)}
\]

definition $I(x_i)$

\[
= E[-\log_2 P(X)]
\]

Example

• Assume
  • $X = \{0, 1\}$
  • $P(X) = \begin{cases} 
  p & X = 0 \\
  1 - p & X = 1 
\end{cases}$

• Then

\[
H(X) = E[I(X)] = -p \log p - (1 - p) \log(1 - p)
\]
Restaurant example

- WillWait response:
  - 6 positive
  - 6 negative

- Entropy

\[
H(x) = -\frac{p}{p+n} \log_2 \frac{p}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n}
\]

\[
= -\frac{6}{6+6} \log_2 \frac{6}{6+6} - \frac{6}{6+6} \log_2 \frac{6}{6+6}
\]

\[
= \log_2 2 = 1
\]

Entropy and tree questions

- Fig. 18.3 has an equal number of positive and negative examples (6 each: p=n=6)
- Training data has entropy of 1 bit:
Tree questions

• Tree questions have a binary response
• Suppose
  • Goal: separate mammals (+) from birds (-)
  • Question: Does it fly?

Animal set:
6 mammals (+), 5 birds (-)

Tree question entropy

• Remember, entropy is: \( E[I(P(X))] = E[-\log_2 P(X)] \)
• For binary categories, we define a short hand:
  • \( q = \frac{p}{p+n} \), the positive rate
  • \( 1 - q = \frac{n}{p+n} \), the negative rate
  • \( B(q) = E[I(P(X))] = -q \log_2 q - (1 - q) \log_2 (1 - q) \)
Tree question entropy

Bird/mammal example

<table>
<thead>
<tr>
<th></th>
<th>p</th>
<th>n</th>
<th>q (+ rate)</th>
<th>B(q)</th>
</tr>
</thead>
<tbody>
<tr>
<td>before question</td>
<td>6</td>
<td>5</td>
<td>6/11</td>
<td>0.99</td>
</tr>
<tr>
<td>¬flies</td>
<td>4</td>
<td>1</td>
<td>4/5</td>
<td>0.72</td>
</tr>
<tr>
<td>flies</td>
<td>2</td>
<td>4</td>
<td>1/3</td>
<td>0.92</td>
</tr>
</tbody>
</table>

Sample computation flies:

\[
b \left( \frac{1}{3} \right) = -\frac{1}{3} \log_2 \left( \frac{1}{3} \right) - \left( 1 - \frac{1}{3} \right) \log_2 \left( 1 - \frac{1}{3} \right) = \frac{1}{3} \log_2 \left( \frac{1}{3} \right) - \left( \frac{2}{3} \right) \log_2 \left( \frac{2}{3} \right) \approx .92
\]

Entropy and tree questions

- Patrons – Categories (None, Some, Full)
  None: 2 examples: \( B(0/2) = 0 \)
  Some: 4 examples: \( B(4/4) = 0 \)
  Many: 6 examples: \( B(2/6) = .918 \)
- Restaurant type (French, Italian, Thai, Burger)
  French: \( B(1/2) = 1 \)
  Italian: \( B(1/2) = 1 \)
  Thai: \( B(2/4) = 1 \)
  Burger: \( B(2/4) = 1 \)
Information gain

• Goal: reduce the amount of information needed to represent the problem

• We can represent the remaining entropy after dividing data into d groups with question A as follows:

\[
\text{Remainder}(A) = \sum_{k=1}^{d} \frac{p_k + n_k}{p + n} B\left(\frac{p_k}{p_k + n_k}\right)
\]

and the information gain as:

\[
\text{Gain}(A) = B\left(\frac{p}{p + n}\right) - \text{Remainder}(A)
\]

Information gain examples

• Mammal/bird flight question
  • Split 11 animals into two groups of size 5 (¬flies) and 6 (flies).

  • Remainder(Does it fly?) = \(\frac{2 + 4}{6 + 5} B\left(\frac{1}{3}\right) + \frac{4 + 1}{6 + 5} B\left(\frac{4}{5}\right) = \frac{6}{11} .92 + \frac{5}{11} .72 \approx .83\)

  • Gain(Does it fly?) = \(B\left(\frac{6}{6 + 5}\right) - \text{Remainder}(\text{Does it fly?}) = 0.99 - 0.83 = 0.16\)
Information gain examples

• Patrons – Categories (None, Some, Full)
  None: 2 examples: \( B(0/2) = 0 \)
  Some: 4 examples: \( B(4/4) = 0 \)
  Many: 6 examples: \( B(2/6) = .918 \)

\[
\text{Gain(Patrons)} = B \left( \frac{6}{6+6} \right) - B \left( \frac{2}{12} \right) = 0 + 4 \cdot 0 + 6 \cdot .918 = .541 \text{ bits}
\]

• Restaurant type (French, Italian, Thai, Burger)
  French: \( B(1/2) = 1 \)
  Italian: \( B(1/2) = 1 \)
  Thai: \( B(2/4) = 1 \)
  Burger: \( B(2/4) = 1 \)

\[
\text{Gain(Type)} = B \left( \frac{6}{6+6} \right) - B \left( \frac{2}{12} \right) = 0 + 4 \cdot 0 + 6 \cdot 1 = 0 \text{ bits}
\]

Decision tree learner

```python
def decision-tree-learner(examples, attributes, parent_examples):
    if empty(examples):
        return plurality-value(parent_examples)  # pick whatever parent had most of
    else if all examples of same class:
        return the class
    else if empty(attributes):  # no more questions to ask
        return plurality-value(examples)
    else:
        a = arg max \text{importance}(a)  # information gain or other measure
        t = new tree(a)  # Create a new tree rooted on most important question
        for each value v associated with a:
            v-examples = \{ e : e \in examples such that e has value v for attribute a \}
            subtree = decision-tree-learner(v-examples, attributes – a, examples)
            t.add_branch(v, subtree)  # Add in new subtree with current value as branch label
        return t
```

Will Indie survive?

• We can build a classifier that predicts if Indiana Jones survives (well, of course he does)
• Possible features:
  • Number of bad guys
  • any snakes?
  • length of Indie’s whip
• Some features might not have much to do with survival:
  • Does Indie have his hat?
  • Did Indie brush his teeth?
Features and overlearning

• Useless features are not good for prediction, but...
  
a learner may pick up on random patterns in the training data and incorporate these into the rules

• Example:
  • Random six sided fair die, learn whether or not we roll 5.
  • Height from which we roll should have any bearing on $P(X=5)=.2$
  • Decision tree may again pick up random patterns, but the lowest classification error rule is to simply say: we will not roll a 5.

Generalization and overfitting

• Learning random patterns that don’t affect function $f$ is called overfitting.

• Overfit models do a great job predicting training data, but do not predict novel data well.

• Decision trees have a tendency to overfit.
Pruning Decision trees

• Overfitting of decision trees is addressed by pruning.

• For each leaf node, we ask ourselves if we had good information gain.
  If the node was informative, we keep it.
  If we didn’t learn anything, we discard.

• NOTE: This is done after the tree is trained.

Pruning decision trees

• How do we know if our decisions were any good?

• Our goal was to separate into the positive and negative classes as well as possible.
Pruning decision trees

• Here we didn’t do a very good job of separating.

• Can we devise a statistic that lets us know if our observed split is statistically **significantly different** from the expected ratio?

\[ \chi^2 \text{ Test} \]

• Suppose decision tree splits a node into \( v \) categories.

• If the node does not add any new information, then we expect each child to have about the same distribution of class labels (e.g. similar % of +/- examples in a 2 class problem).
\( \chi^2 \) Test

- Let’s restrict an example to our two class \((v=2; \) positive and negative\) example.

\[
P(p) = \frac{p}{p+n}, \quad P(n) = \frac{n}{p+n} \quad \text{for the whole data set}
\]

- The question will split the examples into two subsets \( k=1,2 \) with \( p_k \) positive examples and \( n_k \) negative examples.

- How many positive and negatives would we expect if there was no change in distribution from the training data?

\[
\hat{P}_k = \frac{(p_k + n_k)}{p + n}, \quad \hat{n}_k = \frac{(p_k + n_k)}{p + n}
\]

\( \chi^2 \) Test

- We can look at how much our categories differ from what would be expected if the proportion of categories did not change

\[
\chi^2 \text{ test statistic} \quad \Delta = \sum_{k=1}^{d} \left( \frac{p_k - \hat{p}_k}{\hat{p}_k} \right)^2 + \left( \frac{n_k - \hat{n}_k}{\hat{n}_k} \right)^2
\]

where \( d \) is the number of split values

- When \( \Delta \) is small, we are close to the original distribution.
χ² Test

- The test statistic has a distribution that is related to the number of categories – 1. This is referred to as the degrees of freedom (dof) and for a binary classifier, the dof is 2-1=1.

The formula for this is beyond our scope, but the plot shows the probability of having a value of Δ assuming that the distributions are identical.

Cumulative density function (CDF)

- Suppose we integrate P(X|dof) up to Delta

The formula for χ² is beyond scope, but the plot shows the probability of having a value of Δ assuming that the distributions are identical.
χ² test example

Let’s return to the bird/mammal example:

<table>
<thead>
<tr>
<th></th>
<th>p</th>
<th>n</th>
<th>q (+ rate)</th>
<th>B(q)</th>
</tr>
</thead>
<tbody>
<tr>
<td>before question</td>
<td>6</td>
<td>5</td>
<td>6/11</td>
<td>0.99</td>
</tr>
<tr>
<td>¬flies</td>
<td>4</td>
<td>1</td>
<td>4/5</td>
<td>0.72</td>
</tr>
<tr>
<td>flies</td>
<td>2</td>
<td>4</td>
<td>1/3</td>
<td>0.92</td>
</tr>
</tbody>
</table>

Expected in each split if the distribution does not change:

\[
\hat{\rho}_k = p \times \frac{p_k + n_k}{p + n}, \quad \hat{n}_k = n \times \frac{p_k + n_k}{p + n}
\]

\[
\hat{\rho}_{\text{flies}} = \frac{6 \times 2 + 4}{6 + 5} = \frac{36}{11}, \quad \hat{n}_{\text{flies}} = \frac{5 \times 2 + 4}{11} = \frac{30}{11}
\]

\[
\hat{\rho}_{-\text{flies}} = \frac{6 \times 4 + 1}{11} = \frac{30}{11}, \quad \hat{n}_{-\text{flies}} = \frac{5 \times 4 + 1}{11} = \frac{25}{11}
\]

χ² test example

• Compute χ² statistic ∆

\[
\Delta = \sum_k \left( \frac{(p_k - \hat{\rho}_k)^2}{\hat{\rho}_k} + \frac{(n_k - \hat{n}_k)^2}{\hat{n}_k} \right)
\]

\[
\Delta = \frac{(4 - \frac{36}{11})^2}{\frac{36}{11}} + \frac{(1 \frac{30}{11})^2}{\frac{30}{11}} + \frac{(1 \frac{30}{11})^2}{\frac{30}{11}} + \frac{(2 \frac{25}{11})^2}{\frac{25}{11}}
\]

\[
\approx 1.868 + 2.342 + 2.101 + 1.345
\]

\[
\approx 7.55
\]
\(\chi^2\) test example

- We have one dof and \(\Delta=7.52\). Significant change in distributions?
  - 0.994 of 1 dof problems without significant change have \(\Delta<7.52\)
  - About a 0.006 chance that this is not significantly different.
  - Good idea to keep this split

- In general:
  - Define an acceptable level of error (e.g. the 0.006) called a p-value.
  - Very common to use \(p=0.05\)
  - Look up the \(\chi^2\) inverse cdf for the desired p-value.
  - Compute the \(\chi^2\) statistic.
  - Prune if the split does not present a significant difference

\(\chi^2\) Test

- Python does not have \(\chi^2\) routines, but the Scientific Python library does.
  
  ```python
  import scipy.stats.chi2
  dof = 1
  p05 = scipy.stats.chi2.ppdf(.95, dof) # inverse CDF: 3.84
  
  Caveat: Only try this on leaf nodes of a constructed tree!
  - Sometimes, multiple levels have more power than a single one.
  - Pruning as we go can prevent us from ever seeing this.
More thoughts on decision trees

• Continuous/integer-valued attributes
  • Don’t create infinite branches
  • Select a split point
    • Sort values
    • Keep running total of number of +/- examples for each point in sorted list and pick the separating point that gives the best separation.

• See text for information on multivalued attributes and continuous-valued outputs.

Decision tree summary

Relatively straight-forward learners that
• recursively partition the feature space into hyperplanes,
• are sensitive to overtraining, but have methods to prune,
• and are easy for humans to understand
Do I have a good hypothesis function?

- Assume data are *independent* and *identically distributed* (iid)
  - Independent – Examples $e_j = (x_j, y_j)$, $e_k = (x_k, y_k)$ are unrelated to one another when $j \neq k$.
    \[ P(E_j | E_k) = P(E_j) \]
  - Identically distributed – Whatever process generated $e_j$ is also responsible for generating $e_k$ and did not change.

Warning: iid assumptions do not always hold!

Do I have a good hypothesis function?

- We cross-validate the learner on a separate validation set

- Problem: We don’t exploit all our data
k-fold cross validation

A
B
C
D
E

Train

Test

k=3

Extreme case: leave-one-out cross validation (aka jackknife)
k=N

Model selection

- More complex models (e.g. more nodes in a decision tree) learn the training data better, but are they really better?
- For this, we look at validation error

Note: There are also statistics that can help us select models (beyond our scope)
Loss

• Loss functions are a form of utility function that provide a cost for misclassification

\[ L(x, y, \hat{y}) = \text{cost}(\text{predicting } h(x) = \hat{y} \text{ given } f(x) = y) \]

• Suppose that it so useful to find a right whale that we do not mind misclassifying a bunch of non right whales as whales

\[
\begin{align*}
L(x, y = \text{right whale}, \hat{y} = \text{other}) &= 10 \\
L(x, y = \text{right whale}, \hat{y} = \text{right whale}) &= 0 \\
L(x, y = \text{other}, \hat{y} = \text{right whale}) &= 1 \\
L(x, y = \text{other}, \hat{y} = \text{other}) &= 0
\end{align*}
\]
Loss

• Some learners attempt to minimize loss

• Common loss functions

\[ L_1(x, y, \hat{y}) = |y - \hat{y}| \] absolute loss function

\[ L_2(x, y, \hat{y}) = (y - \hat{y})^2 \] squared loss function

\[ L_{0/1}(x, y, \hat{y}) = \begin{cases} 0 & y = \hat{y} \\ 1 & \text{otherwise} \end{cases} \] 0/1 loss function

Generalization loss

• What is our loss when we use a novel data set ε?

• The expected loss requires the distribution of \((X,Y)\) which we probably do not have:

\[ \text{GenLoss}_L(h) = \sum_{(x,y) \in \epsilon} L(x, y, h(x))P(x, y) \]

but we can estimate it empirically on a finite set of examples \(E\) of \(N\) samples:

\[ \text{EmpLoss}_L(h) = \frac{1}{N} \sum_{(x,y) \in E} L(x, y, h(x)) \]

Note: Generalization loss is frequently referred to as risk
Generalization loss

• Selection of our learner $h^*$ now becomes:

$$h^* = \arg \min_{h \in H} EmpLoss_{L,E}(h)$$

• Are we guaranteed $h^* = f$? No:
  • Unrealizability: $f$ may not be in $H$
  • Variance: Learners return different $f$’s for different training sets
  • Noise:
    • $f$ may be noisy (e.g. stochastic component – different $y$’s for the same $x$)
    • The training samples may have mis-measured attributes or incorrect labels
    • Might not have measured important attributes.
  • Complexity: Learner may not achieve a global minimum.

Regularization

• Occam’s Razor states less complex models are better.

• Can we incorporate this into our model selection?

$$Cost(h) = EmpLoss(h) + \lambda \text{Complexity}(h)$$

$$h^* = \arg \min_{h \in H} Cost(h)$$

• The cost function is called a regularization function

Complexity models are beyond our scope, but if you want to know more read about MDL in chapter 20 or information criteria (e.g. AIC,BIC)
Reducing model complexity

- Learner complexity can be reduced by pruning the feature space:
  - feature selection
  - principle components analysis
  - nonlinear dimension reduction

Linear models

- Can be used for classification & regression
- We will start with
  - univariate
  - regression
Univariate linear regression

• Linear combinations of weights $w_j$ with input $x$ to predict $y$:

$$ y = w_1 x + w_0 $$

• Objective: Find vector $\hat{w}$ that minimizes a loss function

$$ Loss(h_w) = \sum_{j=1}^N L_2(y_j, h_w(x_j)) $$

$$ = \sum_{j=1}^N (y_j - h_w(x_j))^2 $$

$$ = \sum_{j=1}^N (y_j - (w_1 x_j + w_0))^2 $$

Minimizing L2 loss

• We want to find $w^* = \arg \min_w Loss(h_w)$

• Derivative of loss function is convex $\rightarrow$ single global minimum
Minimizing L2 loss

• Set derivatives of both unknowns to 0

\[
\frac{\partial}{\partial w_0} \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2 = 0
\]

\[
\sum_{j=1}^{N} 2(y_j - (w_1 x_j + w_0)) \frac{\partial}{\partial w_0} (y_j - (w_1 x_j + w_0)) = 0
\]

\[
\frac{\partial}{\partial w_1} \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2 = 0
\]

\[
\sum_{j=1}^{N} (y_j - (w_1 x_j + w_0)) = \frac{0}{2}
\]

\[
\sum_{j=1}^{N} y_j - w_1 \sum_{j=1}^{N} x_j - N w_0 = 0
\]

\[
w_0 = \frac{\sum_{j=1}^{N} y_j - w_1 \sum_{j=1}^{N} x_j}{N}
\]

Minimizing L2 loss

• We now have 2 equations in \( w_0 \) and \( w_1 \), and can use algebra to solve:

\[
w_0 = \frac{\sum_{j=1}^{N} y_j - w_1 \sum_{j=1}^{N} x_j}{N}
\]

\[
w_1 = \frac{N \sum_{j=1}^{N} x_j y_j - \left( \sum_{j=1}^{N} x_j \right) \left( \sum_{j=1}^{N} y_j \right)}{N \sum_{j=1}^{N} x_j^2 - \left( \sum_{j=1}^{N} x_j \right)^2}
\]
Beyond linear models

• Not all loss functions have closed form solutions
• In these cases, we return to what we learned about optimization and use a gradient descent method

```python
def gradient_descent(w):
    while not converged:
        for i in 1:len(w):
            w[i] = w[i] - \alpha \frac{\partial}{\partial w_i} Loss(w)
```

• \( \alpha \) is the learning rate
  • We called this the step size in optimization
  • Can be fixed, decay over time, or be adaptively set

Beyond linear models

• In general for vector \( \mathbf{w} \)

\[
\frac{\partial}{\partial w_i} \text{Loss}(\mathbf{w}) = \frac{\partial}{\partial w_i} (y - h_{\mathbf{x}}(x))^2 \\
= 2(y - h_{\mathbf{x}}(x)) \frac{\partial}{\partial w_i} (y - h_{\mathbf{x}}(x)) \\
= 2(y - h_{\mathbf{x}}(x)) \frac{\partial}{\partial w_i} (y - (w_i x + w_0))
\]

(here we show a simple linear case for \( h_{\mathbf{x}}(x) \))

which yields:

\[
\frac{\partial}{\partial w_i} \text{Loss}(\mathbf{w}) = 2(y - h_{\mathbf{x}}(x)) \frac{\partial}{\partial w_i} (y - (w_i x + w_0)) \\
= -2(y - h_{\mathbf{x}}(x)) x
\]
Beyond linear models

• Remember the gradient descent update rule

\[ w_j = w_j - \alpha \frac{\partial}{\partial w_j} \text{Loss}(w) \]

• The update rules for the previous example are:

\[ w_0 = w_0 - \alpha \frac{\partial}{\partial w_0} \text{Loss}(\tilde{w}) \]
\[ w_1 = w_1 - \alpha \frac{\partial}{\partial w_1} \text{Loss}(\tilde{w}) \]
\[ = w_0 + \alpha \frac{2}{2} (y - h_0(x)) \]
\[ = w_1 + \alpha \frac{2}{2} (y - h_1(x)) \]
\[ \text{we fold the 2 into } \alpha^* \]

• To minimize the sum of losses across \((x_j, y_j)\):

\[ w_0 = w_0 + \alpha^* \sum_{j=1}^{N} (y_j - h_0(x_j)) \]
\[ w_1 = w_1 + \alpha^* \sum_{j=1}^{N} (y_j - h_1(x_j)) x_j \]

while \( \alpha^* = 2 \alpha \), we will frequently just write \( \alpha \)

Gradient descent types

• Batch (offline) gradient descent
  • w’s update based on all samples
  • convergence guaranteed
    with small \( \alpha \)
  • may be slow

\[ w_i = w_i - \alpha \sum_{j=1}^{N} \frac{\partial}{\partial w_j} \text{Loss}(w, y_j, x_j) \]

• Stochastic gradient descent
  • update w after every example (good for online learning)
  • usually faster
  • No guarantee of convergence with fixed \( \alpha \) (may oscillate about a minimum)

\[ w_i = w_i - \alpha \frac{\partial}{\partial w_j} \text{Loss}(w, y_j, x_j) \]

mini batch – hybrid between two is popular!
Multivariate linear regression

• Observations are multidimensional:
• Weight vector has a component for each \( x_{ji} \):

\[
h_{\text{mw}}(\vec{x}_i) = w_0 + w_1 x_{i1} + w_2 x_{i2} + \cdots + w_D x_{iD} = w_0 + \sum_{j=1}^{D} w_j x_{ij}
\]

• We frequently eliminate \( w_0 \)'s special case:

  Redefine \( \vec{x}_i \) as \([1, x_{i1}, x_{i2}, \ldots, x_{iD}]\), then

\[
h_{\text{mw}}(\vec{x}_i) = \sum_{j=0}^{D} w_j x_{ij} = w^T \vec{x}_i \text{(product of } 1 \times D \text{ and } D \times 1 \text{ vectors)}
\]

Multivariate linear regression

• As before, we minimize the L2 loss.
• Gradient descent is very similar to the univariate case:

\[
w_i = w_i + \alpha \sum_{j=1}^{N} (y_j - h_{\text{mw}}(x_j)) x_{ij}
\]

• However for linear functions we can solve analytically for a closed-form equation that does not require iteration

\[
w^* = (X^T X)^{-1} X^T y
\]

\[
X = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1D} \\ 1 & x_{21} & x_{22} & \cdots & x_{2D} \\ 1 & \cdots & \cdots & \cdots & \cdots \\ 1 & x_{D1} & x_{D2} & \cdots & x_{DD} \end{bmatrix}, 
\]

\[
y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}
\]
Overfit regressions

• Not a problem for univariate linear regression
• Problematic for multivariate
• Regularization provides penalties for increasing complexity

\[
\text{Cost}(h_w) = \text{EmpLoss}(h) + \lambda \text{Complexity}(h)
\]

\[
\text{Complexity}(h_w) = L_q(w) = \sum_i |w_i|^q
\]

we regularize by picking the minimal cost hypothesis.

Regularization of regression

\( L_1 \) tends to produce sparse models with many zero weights.

- Minimizing Cost is equivalent to minimizing loss with constraint that complexity \( \leq \) some constant.
- Complexity increases as \( w^* \) moves away from the origin

Isolines for loss function, loss is lowest at center

L1 complexity constraint is more likely to intersect near an axis.
Linear classification

- Regression lines can be used to classify examples
- We look for a linear separating line (hyperplane when data $\in \mathbb{R}^3$ or higher)

\[ \mathbf{w} \cdot \mathbf{x} = \sum_{j=1}^{D} w_j x_j \]

Linear classification

- Suppose we pick a vector $\mathbf{w}$ perpendicular to the decision boundary
- Consider the dot product between $\mathbf{w}$ and an arbitrary point
Geometric interpretation of the dot product

\[ \theta_1 < 90^\circ \Rightarrow \cos(\theta_1) > 0 \]
\[ \theta_2 > 90^\circ \Rightarrow \cos(\theta_2) < 0 \]

Linear classification

• Once we have \( \mathbf{w} \), we can classify by taking the dot product and looking at the sign.

\[ h_w(x) = \text{Threshold}(w \cdot x) \]

\[ \text{Threshold}(z) = \begin{cases} 
1 & z \geq 0 \\
-1 & z < 0 
\end{cases} \]

• How do we choose \( \mathbf{w} \)?
  • Fisher provides a method to select without iterating, but it won’t be useful for later applications...
  • It turns out a variant of the gradient descent rule used for regression is applicable.

Recall the regression rule

\[ w_i = w_i + \alpha (y - h_w(x)) x_i \]
Linear classification – Perceptron learning rule

• Unlike the regression rule, we are not trying to minimize squared error, but loss.
• We only apply the update when we would have gotten this wrong

```python
def perceptron_learning(w, x, y):
prediction = threshold(w x)
if y != prediction:
    # Oops! Need w x larger if y =1.
    # Increase w when x positive,
    # Decrease when x negative
    # Opposite when w x = -1
    w = w + a y x
```

Perceptron learning rule

• Iterate through data (1 iteration = 1 epoch) and repeat until no errors
• Convergence may be slow, but guaranteed for linearly separable data.

• Most data sets are not linearly separable.
• When non linearly separable examples are presented in random order, we will converge to a stable classifier if a decays linearly with epoch number.
Linear classifier with logistic regression function

- Hard vs. soft thresholds

\[ h_w(x) = \text{Logistic}(w \cdot x) = \frac{1}{1 + e^{-w \cdot x}} \]

- Has nicer properties (e.g. differentiable)

Learning with logistic regression

- Remember that our update rule was:  
  \[ w_i = w_i - \alpha \frac{\partial}{\partial w_i} \text{Loss}(w) \]

- When the hard threshold is replaced with logistic regression:
  - It is easier if we use labels \([0,1]\) instead of \([-1,1]\) as this is the range covered by the function
  - The derivative of the loss needs to be recomputed (see text for derivation)
    \[ w_i = w_i - \alpha(y - h_w(x))h_w(x)(1 - h_w(x))x_i \]
  - This results in a smoother and more predictable learning curve
Connectionist networks (artificial neural networks)

- Activations functions for perceptrons are nonlinear:
  - hard threshold
  - logistic regression (frequently called sigmoid function)

- Linking perceptrons together provides complex function modeling capability
Decision boundary capability as a function of network depth

An intuitive view of neural nets

- Suppose we combine two perceptrons whose output functions are reversed

- This could be used to model a ridge in output space
Learning in a neural network

• Consider input vector $\mathbf{x}$
• Output vector $\mathbf{a}$

![Multilayer Feed-forward network](image)

Learning in a neural network

• Similar to the regression problem, for output $\mathbf{a}$ and desired output $\mathbf{y}$, we can find the loss gradient for each output node

$$
\frac{\partial}{\partial \mathbf{w}} \text{Loss}(\mathbf{w}) = \frac{\partial}{\partial \mathbf{w}} [y - h_\mathbf{w}(\mathbf{x})]^2 = \frac{\partial}{\partial \mathbf{w}} \sum_{k=1}^{n} (y_k - a_k)^2 = \sum_{k=1}^{n} \frac{\partial}{\partial \mathbf{w}} (y_k - a_k)^2
$$

$$
a_i = \frac{1}{1 + e^{-\text{output}_i \cdot \text{input}_i}}
$$

and use the perceptron learning rule for the sum of the gradients at the output layer.
Back-propagation

• What should the targets be for the previous input layer?

Back propagating error (overview)

• Error of the kth output: $Err_k = y_k - a_k$

• We can compute the gradient for any input node (in) and apply the regression rule.

• This gives us a new set of weights for the output node.
Back propagating error (overview)

• After applying the update to the output layer, there still exists loss.
• We assign a portion of the loss to each of the input nodes based on their weight.
• This contribution is computed for each node of the current layer.

\[ \sum_i w_i \times \text{Loss}_2 \]

Back propagating error (overview)

• Now we can look at the sum of losses attributable to each node in the previous layer.
• The sum of these provides us with a loss to minimize.
• Repeat recursively.

\[ \sum_i w_{\text{out},1,i} \times \text{Loss}_1 \]

\[ \sum_i w_{\text{out},2,i} \times \text{Loss}_2 \]

\[ \sum_i w_{\text{out},3,i} \times \text{Loss}_3 \]
Concrete example

Example based on Christopher Olah’s blog [post](#).

Activation fn example for backprop

Partial derivatives

\[
\frac{\partial L}{\partial u_{out}} = \frac{1}{2} (y - u_{out})(-1) = u_{out} - y
\]

\[
\frac{\partial u_{out}}{\partial u_{in}} = \sigma(u_{in})(1 - \sigma(u_{in}))
\]

\[
\frac{\partial u_{in}}{\partial u_{x_1}} = w_1 x_1^2
\]

\[
\frac{\partial u_{in}}{\partial u_{x_2}} = w_2
\]

\[
\frac{\partial u_{in}}{\partial u_{x_3}} = x_1^3
\]

\[
\frac{\partial u_{in}}{\partial u_{x_4}} = x_2
\]
Activation fn example for backprop

To update $w_j$ we use the chain rule:

$$\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial u_m} \frac{\partial u_m}{\partial u_i} \frac{\partial u_i}{\partial w_j} = (u_m - y) \sigma(u_m)(1 - \sigma(u_m))x_i^3$$

from previous slide

$$\frac{\partial L}{\partial u_m} = u_m - y$$

$$\frac{\partial u_m}{\partial u_i} = \sigma(u_m)(1 - \sigma(u_m))$$

$$\frac{\partial u_m}{\partial w_j} = x_i^3$$

Concrete example

$$y = 0, w = \begin{bmatrix} .02 \\ .01 \end{bmatrix}, x = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$$

implies

$$u_m = w_1 x_1^3 + w_2 x_2 = .02 \cdot 3^3 + .01 \cdot 5 = .59$$

$$u_m = \frac{1}{1 + e^{-u_m}} = \frac{1}{1 + e^{-0.59}} = .6434$$

$$L = \frac{1}{2} (y - u_m)^2 = .5 \cdot (0 - .6434)^2 = .2070$$

$$\frac{\partial L}{\partial w_1} = .6434 \cdot 27 = 3.981$$

$$\frac{\partial L}{\partial w_2} = .2294 \cdot 27 = 6.233$$
Activation fn example for backprop

\[ u_{out} = \sigma(u_{in}) \]

\[ L = \frac{1}{2}(y - u_{out})^2 \]

\[ u_{out} = w_1 x_1^2 + w_2 x_2 \]

Suppose we have a learning rate \( \varepsilon = .01 \)

Update of \( w_2 \) is left as an exercise, but loss with only \( w_1 \) changed:

\[ y = 0, w = \begin{bmatrix} -.06 \\ .01 \end{bmatrix} \]

implies

\[ u_x = w_1 x_1^2 + w_x x_2 = -.06 \cdot 3^2 + .01 \cdot 5 = -1.57 \]

\[ u_{out} = \frac{1}{1 + e^{-u_x}} = \frac{1}{1 + e^{1.57}} = .1722 \]

\[ L = \frac{1}{2}(y - u_{out})^2 = .5 \cdot (0 - .1722)^2 = .0148 \] old \( L = .2070 \)

---

Neural net summary

- **Supervised learner**
  - Training labels either
    - High value for class (n classes → n output nodes)
    - Encoding of class information
  - Iterative training typically using a gradient descent algorithm (e.g. back propagation)

- **Classification**
  - Present features to input nodes
  - Interpret output nodes for category
Neural net summary

• Disadvantages
  • frequently hard to interpret
  • Many parameters require large data sets
  • Doesn’t do well with imbalanced examples
  • Slow to train
  • Overfits easily and regularization is important

• Advantages
  • Flexible, nonlinear learner
  • Deep architectures are very powerful

Non-parametric models

• Neural nets and decision trees have models with parameters
  • decision node parameters: attribute and cut-point/categories for sub-trees
  • neural nets: weights and connections

• Non-parametric models
  • Cannot be characterized by a bounded set of parameters
  • Simplest case:
    Look at every example and use it to classify a novel example. (Parameters $\propto$ #training examples)
  • Called instance- or memory-based learning
Nearest neighbor models

• Use a distance metric to find the k closest neighbors, e.g. for continuous attributes:

\[
L^p(\tilde{x}_j, \tilde{x}_q) = \left( \sum_{i=1}^{D} |x_{j,i} - x_{q,i}|^p \right)^{1/p}
\]

• Use the plurality of labels that are the k closest

Nearest neighbor models

• The good
  • Simplicity
  • Effective technique for low-dimensional data

• The Bad – Searching is expensive with large training sets, but we can mitigate for this:
  • trees – Similar to a decision tree (split on value, may at times need to search both sides)
  • Locally sensitive hash tables
    • Hash functions
      • set of projections on to lines (similar to linear classification examples)
      • Line projections are discretized into buckets
      • Can be much more effective than tree approach
Nearest neighbor models

- and the Ugly
  - N points uniformly distributed in an \( \mathbb{R}^D \) unit hypercube.
  - To capture \( r = .01 \) of the observations, what edge length \( l \) would we need in a random sample?

- Samples are randomly distributed and total volume is 1, so we need a volume of \( r \) (.01).

\[
l^d = r \rightarrow l = r^{\frac{1}{D}}
\]

- \( d = 1 \rightarrow l = .01^{\frac{1}{1}} = .01 \)
- \( d = 10 \rightarrow l = .01^{\frac{1}{10}} = .63 \)
- \( d = 100 \rightarrow l = .01^{\frac{1}{100}} = .96 \)

The curse of dimensionality!

As the dimension grows, the size of each edge on the hypercube grows as well!

Support vector machines (SVMs)

- A margin is the distance to the closest examples on either side of a hyperplane.
- SVM approaches attempt to maximize the margin
Support vector machines

- Can only separate linear problems, but a kernel function can project the data into a higher dimensional space where perhaps the data can be better separated.

Support vector machines

- Maximal margins are computed as functions of training examples.
- Consequently
  - SVMs are nonparametric techniques.
  - In practice, only a small subset of training examples, the support vectors, are required.
- The training algorithm is beyond our scope, but is essentially an optimization problem.
Bias and variance

- Error in learning comes from two sources: bias and variance

![Graph showing Mean Error vs. Model Complexity](image)

- Bias – Large when learners make consistently incorrect predictions
- Variance – Large when different training sets result in different predictions

Ensemble learning

- Ensemble learners frequently are a collection of weak learners that are combined to form a robust classifier

  *Weak learner* – A simple learning algorithm that is likely to have a high bias (e.g. a single node, or stump, of a decision tree)

- Ensemble learners typically use collections of weak classifiers to reduce both bias and variance.
Adaptive Boosting (ADABOOST)

- Type of ensemble learning algorithm
- Use decision stumps as the weak learner
- Examples are weighted. Loss is greater for examples with higher weight

Adaptive boosting

- Start with uniform weights
- Learn the decision tree stump
  - Redistribute weights: misclassified training examples get more weight
  - Produce a classification weight as a function of error
  - Iterate until k learners are produced
Adaptive boosting

- Classification
  - Classify an example by each of the $k$ weak learners
  - Use plurality of weighted decisions

- A very interesting tidbit...
  Letting $k$ grow beyond the point of having all training examples lets ADABOOST frequently continue to improve generalization scores.

Some interpret this as ADABOOST being robust to overtraining.