Machine Learning

Study of algorithms that
- improve their performance
- at some task
- with experience

Data → Machine Learning → Understanding

Supremacy of Machine Learning

- Machine learning is preferred approach to
  - Speech recognition, Natural language processing
  - Web search – result ranking
  - Medical outcomes analysis
  - Robot control
  - Computational biology
  - Sensor networks
  - ...

- This trend is accelerating
  - Improved machine learning algorithms
  - Improved data capture, networking, faster computers
  - Software too complex to write by hand
  - New sensors / I/O devices
  - Demand for self-customization to user, environment

Reinforcement Learning

- Robotic control
  - helicopter maneuvering, autonomous vehicles
  - Mars rover - path planning, oversubscription planning
  - elevator planning
- Game playing - backgammon, tetris, checkers
- Neuroscience
- Computational Finance, Sequential Auctions
- Assisting elderly in simple tasks
- Spoken dialog management
- Communication Networks – switching, routing, flow control
- War planning, evacuation planning

Exam

- Much like midterm, but a bit easier
- Will include one problem from midterm
- Will also include
  - Unsupervised learning
  - Reinforcement learning
  - Instance-based learning
Planning Agent

Environment

Static vs. Dynamic

Fully vs. Partially Observable

Deterministic vs. Stochastic

Perfect vs. Noisy

Deterministic vs. Stochastic

Instantaneous vs. Durative

Percepts

Actions

What action next?

Bellman Equations for MDP

- $<S, A, P, R, s_0, \gamma>$
- Define $V^*(s)$ (optimal value) as the maximum expected discounted reward from this state.
- $V^*$ should satisfy the following equation:

$$V^*(s) = \max_{a \in A(s)} \sum_{s' \in S} P(s'|s, a) [R(s, a, s') + \gamma V^*(s')]$$

Summary RL

- Bellman Equation
- Value iteration
- Credit assignment problem
- Exploration / exploitation tradeoff
  - Greedy in limit of infinite exploration
  - Optimistic exploration

Space of ML Problems

Type of Supervision (eg, Experience, Feedback)

<table>
<thead>
<tr>
<th>What’s Being Learned?</th>
<th>Labeled Examples</th>
<th>Reward</th>
<th>Nothing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discrete Function</td>
<td>Classification</td>
<td>Clustering</td>
<td></td>
</tr>
<tr>
<td>Continuous Function</td>
<td>Regression</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Policy</td>
<td>Apprenticeship Learning</td>
<td>Reinforcement Learning</td>
<td></td>
</tr>
</tbody>
</table>

Generalization

- Hypotheses must *generalize* to correctly classify instances not in the training data.
- Simply memorizing training examples is a consistent hypothesis that *does not generalize*.
Learning as function approximation

- What's a **good** approximation?

### Overfitting

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>On training data</th>
<th>On test data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Model complexity (e.g., number of nodes in decision tree)

© Daniel S. Weld

---

Learning as Optimization

- **Methods**
  - Closed form
  - Greedy search
  - Gradient ascent

- **Loss Function**
  - Minimize **loss** over training data (test data)
  - Loss(h, data) = error(h, data) + complexity(h)
  - Error + regularization

---

Bia / Variance Tradeoff

- **Variance:** $E[(h(x^*) - h(x^*))^2]$  
  How much $h(x^*)$ varies between training sets  
  Reducing variance risks underfitting

- **Bias:** $[h(x^*) - f(x^*)]$  
  Describes the **average** error of $h(x^*)$  
  Reducing bias risks overfitting

---

Regularization

Regularization: $E_{RMS}$ vs. $\ln \lambda$

---

![Graph of model complexity vs. accuracy](image1)

![Graph of variance and bias](image2)

![Graph of regularization](image3)
Machine Learning
- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning
  - Parametric
  - Non-parametric
  - Y Continuous
  - Y Discrete
  - Gaussians
    - Learned in closed form
  - Linear Functions
    - 1. Learned in closed form
    - 2. Using gradient descent
- Decision Trees
  - Greedy search; pruning
- Probability of class | features
  - 1. Learn P(Y), P(X|Y); apply Bayes
  - 2. Learn P(Y|X) w/ gradient descent
- Non-probabilistic Linear Classifier
  - Learn w/ gradient descent

Probabilities
- Random variables, distributions
- Axioms of probability
- Marginal, joint & conditional probabilities
- Sum rule, product rule, Bayes rule
- Independence, conditional independence

Our Favorite Distributions

<table>
<thead>
<tr>
<th>Discrete</th>
<th>Continuous</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary {0, 1}</td>
<td>M Values</td>
</tr>
<tr>
<td>Single Event</td>
<td>Bernoulli</td>
</tr>
<tr>
<td>Sequence</td>
<td>Binomial</td>
</tr>
<tr>
<td>(N trials)</td>
<td>Multinomial</td>
</tr>
<tr>
<td>Conjugate</td>
<td>Beta</td>
</tr>
<tr>
<td>Prior</td>
<td>Dirichlet</td>
</tr>
</tbody>
</table>

Inference

<table>
<thead>
<tr>
<th>Prior</th>
<th>Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform</td>
<td>The most likely</td>
</tr>
<tr>
<td>Any</td>
<td>The most likely</td>
</tr>
<tr>
<td>Any</td>
<td>Weighted combination</td>
</tr>
</tbody>
</table>

Learning Gaussian Parameters

$$M L E: \begin{align*}
\hat{\mu}_{M L E} &= \frac{1}{N} \sum_{i=1}^{N} x_i \\
\hat{\sigma}^2_{M L E} &= \frac{1}{N} \sum_{i=1}^{N} (x_i - \hat{\mu})^2
\end{align*}$$

Linear Regression

$$\hat{h}_w(x) = w_1x + w_0$$

$$w_1 = \frac{N \sum (x_i y_i) - (\sum x_i)(\sum y_i)}{N \sum (x_i^2) - (\sum x_i)^2}$$

$$w_0 = (\sum y_i) - w_1 (\sum x_i) / N$$
Bayesian Learning

- Let set of categories be \( \{c_1, c_2, \ldots, c_n\} \)
- Let \( E \) be description of an instance.
- Determine category of \( E \) by determining for each \( c_i \)

Use Bayes rule:

\[
P(c_i | E) = \frac{P(c_i)P(E | c_i)}{P(E)}
\]

Optimal classification

- Theorem: Bayes classifier \( h_{\text{Bayes}} \) is optimal!

\[
error_{\text{true}}(h_{\text{Bayes}}) \leq error_{\text{true}}(h), \quad \forall h
\]

Why?

\[
p_h(error) = \int p_h(error | x)p(x) = \int \int \delta(h(x), y)p(y|x)p(x)
\]

Naïve Bayes

- Naïve Bayes assumption:
  - Features are independent given class:
  \[
P(X_1, X_2 | Y) = P(X_1 | Y)P(X_2 | Y)
\]
  - More generally:
  \[
P(X_1, X_2 | Y) = \prod P(X_i | Y)
\]

- How many parameters now?
  - Suppose \( X \) is composed of \( n \) binary features

Bag of Words Approach

What if we have continuous \( X_i \)?

E.g., character recognition: \( X_i \) is \( i^{th} \) pixel.

Gaussian Naïve Bayes (GNB):

\[
P(X_i = x | Y = y_k) = \frac{1}{\sigma_{ik}\sqrt{2\pi}} e^{-\frac{(x - \mu_{ik})^2}{2\sigma_{ik}^2}}
\]

Sometimes assume variance
- is independent of \( Y \) (i.e., \( \sigma_{ik} \)),
- or independent of \( X_i \) (i.e., \( \sigma_{ik} \)),
- or both (i.e., \( \sigma_{ik} \))

<table>
<thead>
<tr>
<th>sandbear</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>about</td>
<td>2</td>
</tr>
<tr>
<td>all</td>
<td>2</td>
</tr>
<tr>
<td>Africa</td>
<td>1</td>
</tr>
<tr>
<td>apple</td>
<td>0</td>
</tr>
<tr>
<td>anxious</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>gas</td>
<td>1</td>
</tr>
<tr>
<td>oil</td>
<td>1</td>
</tr>
<tr>
<td>Zaire</td>
<td>0</td>
</tr>
</tbody>
</table>

Naïve Bayes vs. Logistic Regression

Learning: \( h: X \mapsto Y \)

- \( X \) – features
- \( Y \) – target classes

<table>
<thead>
<tr>
<th>Generative</th>
<th>Discriminative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assume functional form for ( P(X</td>
<td>Y) ) ( \Rightarrow ) assume cond indep</td>
</tr>
<tr>
<td>( P(Y) )</td>
<td>( \Rightarrow ) Est params from training data</td>
</tr>
<tr>
<td>Gaussian NB for cont features</td>
<td>Gaussian NB for cont features</td>
</tr>
<tr>
<td>Bayes rule to calc. ( P(Y</td>
<td>X = x) )</td>
</tr>
<tr>
<td>Indirect computation</td>
<td>Directly calculate ( P(Y</td>
</tr>
<tr>
<td>Can also generate a sample of the data</td>
<td>Can’t generate data sample</td>
</tr>
</tbody>
</table>
Logistic w/ Initial Weights

\[ w_0 = 20, \quad w_1 = -5, \quad w_2 = 10 \]

\[ \text{Loss}(H_w) = \text{Error}(H_w, \text{data}) \]

Minimize error \( \Rightarrow \) Maximize \( \ell(w) = \ln P(D_y | D_x, H_w) \)

Loss function:

\[
\ell(w) = -\sum_{i=1}^{N} \left[ y_i \log(\sigma(x_i^T w)) + (1 - y_i) \log(1 - \sigma(x_i^T w)) \right]
\]

Update rule:

\[
\Delta w = \eta \nabla \ell(w)
\]

\[
w_i^{(t+1)} = w_i^{(t)} + \eta \frac{\partial \ell(w)}{\partial w_i}
\]

Binary Perceptron Algorithm

- Start with zero weights
- For each training instance \((x,y^*)\):
  - Classify with current weights
  - If correct (i.e., \(y = y^*\)), no change!
  - If wrong: update

\[
w = w + y^* \cdot f
\]

Three Views of Classification

- Naive Bayes:
  - Parameters: from data statistics
  - Parameters: probabilistic interpretation
  - Training: one pass through the data
- Logistic Regression:
  - Parameters from gradient ascent
  - Parameters: linear, probabilistic model, and discriminative
  - Training: one pass through the data per gradient step, use validation to stop
- The perceptron:
  - Parameters from reactions to mistakes
  - Parameters: discriminative

Hypotheses: decision trees \( f : X \rightarrow Y \)

- Each internal node tests an attribute \(x_i\)
- Each branch assigns an attribute value \(x_i \rightarrow \cdot\)
- Each leaf assigns a class \(y\)
- To classify input \(x\)? traverse the tree from root to leaf, output the labeled \(y\)

What functions can be represented?

\[
cyl=3 \lor (cyl=4 \land (\text{maker}=\text{asia} \lor \text{maker}=\text{europe})) \lor \ldots
\]

Two Questions

- Which attribute gives the best split?
- When to stop recursion?
Which attribute gives the best split?
Many answers (accuracy, misclassification rate, etc). 
Most common method is:
“Attribute with the highest information gain, IG”

\[ IG(X) = H(Y) - H(Y | X) \]

\[
H(Y) = - \sum_{y=1}^{k} P(Y = y) \log_2 P(Y = y)
\]

\[
H(Y | X) = - \sum_{j=1}^{k} \sum_{y=1}^{k} P(X = x_j, Y = y) \log_2 P(Y = y | X = x_j)
\]

Reduced Error Pruning
Split data into training & validation sets (10-33%)
Train on training set (overfitting)
Do until further pruning is harmful:
1) Evaluate effect on validation set of pruning each possible node (and tree below it)
2) Greedily remove the node that most improves accuracy of validation set

Ensembles of Classifiers
- Traditional approach: Use one classifier
- Can one do better?
- Approaches:
  - Cross-validated committees
  - Bagging
  - Boosting
  - Stacking

Ensembles of Classifiers
- Assume
  - Errors are independent (suppose 30% error)
  - Majority vote
- Probability that majority is wrong…
  = area under binomial distribution
  - If individual area is 0.3
  - Area under curve for \( \geq 11 \) wrong is 0.026
  - Order of magnitude improvement!

Fighting the bias-variance tradeoff
- Simple (a.k.a. weak) learners are good
  - e.g., naive Bayes, logistic regression, decision stumps (or shallow decision trees)
  - Low variance, don’t usually overfit
- Simple (a.k.a. weak) learners are bad
  - High bias, can’t solve hard learning problems
- Can we make weak learners always good???
  - No!!
  - But often yes…

Boosting
- Idea: given a weak learner, run it multiple times on (rewighted) training data, then let learned classifiers vote
- On each iteration \( t \):
  - weight each training example by how incorrectly it was classified
  - Learn a hypothesis \( h_t \)
  - A strength for this hypothesis \( a_t \)
- Final classifier:
  \[
  h(x) = \text{sign} \left( \sum_t a_t h_t(x) \right)
  \]
Machine Learning
- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning
- Parametric
- Non-parametric
- Nearest neighbor
- Kernel density estimation
- Support vector machines

k-Nearest Neighbor
Instance-based learning, four things to specify:
1. A distance metric
   Euclidian (and many more)
2. How many nearby neighbors to look at?
   k
3. A weighting function (optional)
   Unused
4. How to fit with the local points?
   Return the average output
   predict: \( \frac{1}{k} \sum y_i \) (summing over k nearest neighbors)

Kernel Regression
Instance-based learning:
1. A distance metric
   Euclidian (and many more)
2. How many nearby neighbors to look at?
   All of them
3. A weighting function
   \( w_i = \exp(-D(x_i, \text{query})^2 / K_w^2) \)
   Nearby points to the query are weighted strongly,
   Far points weighted weakly.
   The \( K_w \) parameter is the Kernel Width. Very important.
4. How to fit with the local points?
   Predict the weighted average of the outputs:
   predict = \( \frac{\sum w_i y_i}{\sum w_i} \)

Support Vector Machines
- Key insight
  - Max Margin
- Clever trick
  - Kernel trick

Linear Separators
- Which of these linear separators is optimal?
Support Vector Machines

- Maximizing the margin:
  - good according to intuition, theory, practice
- SVMs find separator with max margin
  - Convex optimization
  - Quadratic programming (off the shelf solns)
- Reduced set of features!

\[ \min_{w} \frac{1}{2}||w||^2 \]
\[ y_i (w \cdot f(x_i)) \geq 1 \]

Support Vectors:
- data points on the canonical lines
Non-support Vectors:
- everything else
- moving them will not change \( w \)

What if the data is not linearly separable?

Add More Features!!!

2D \rightarrow 3D, using new features: \( F(x) = (x_1^2, x_2^2, \sqrt{2} x_1 x_2) \)

Dual SVM Formulation

Derivation requires computing Lagrangian & some advanced math

Notes:
- One \( \alpha \) for each training example
- Sums over all training examples
- dot product
- scalars

Kernel trick:
- Can compute \( F(x)F(x') \) without computing \( F(x) \) or \( F(x') \) in many cases

Overfitting?

- Huge feature space with kernels, what about overfitting???
  - Maximizing margin leads to sparse set of support vectors
  - Some interesting theory says that SVMs search for simple hypothesis with large margin
  - Often robust to overfitting
    - But everything overfits sometimes!!!
    - Can control by choice of Kernel

Machine Learning

- Supervised Learning
- Reinforcement Learning
- Unsupervised Learning
- Parametric
- Non-parametric

Agglomerative Clustering
- K-means
- Expectation Maximization (EM)
- Principle Component Analysis (PCA)
**Example: K-Means for Segmentation**

- K=2
- K=3
- Original

**Agglomerative Clustering**

- **Agglomerative clustering:**
  - First merge very similar instances
  - Incrementally build larger clusters out of smaller clusters

- **Algorithm:**
  - Maintain a set of clusters
  - Initially, each instance in its own cluster
  - Repeat:
    - Pick the two closest clusters
    - Merge them into a new cluster
    - Stop when there's only one cluster left

- Produces not one clustering, but a family of clusterings represented by a dendrogram

**Agglomerative Clustering**

- How should we define “closest” for clusters with multiple elements?

- Many options:
  - Closest pair (single-link clustering)
  - Farthest pair (complete-link clustering)
  - Average of all pairs
  - Ward’s method (min variance, like k-means)

- Different choices create different clustering behaviors

**K-Means**

- An iterative clustering algorithm
  - Pick K random points as cluster centers (means)
  - Alternate:
    - Assign data instances to closest mean
    - Assign each mean to the average of its assigned points
  - Stop when no points' assignments change

**EM**

- Another iterative clustering algorithm
  - Pick K random cluster models
  - Alternate:
    - Assign data instances proportionately to different models
    - Revise each cluster model based on its proportionately assigned points
  - Stop when no changes
Prefered on Cluster Sizes

Feature Selection

- Want to learn \( f : X \rightarrow Y \)
  - \( X = \langle X_1, \ldots, X_n \rangle \)
  - but some features are more important than others

- Approach: select subset of features to be used by learning algorithm
  - Score each feature (or sets of features)
  - Select set of features with best score

Linear projection and reconstruction

Basic PCA algorithm

- Start from \( m \) by \( n \) data matrix \( X \)
- Recenter: subtract mean from each row of \( X \)
  - \( X_c \leftarrow X - \bar{X} \)
- Compute covariance matrix:
  - \( \Sigma \leftarrow \frac{1}{m} X_c^T X_c \)
- Find eigen vectors and values of \( \Sigma \)
- Principal components: \( k \) eigen vectors with highest eigen values

Machine Learning

Co-Training Motivation

- Learning methods need labeled data
  - Lots of \( \langle x, f(x) \rangle \) pairs
  - Hard to get… (who wants to label data?)

- But unlabeled data is usually plentiful…
  - Could we use this instead??????

- Semi-supervised learning
Co-training

Suppose

- Have little labeled data + lots of unlabeled
- Each instance has two parts:
  \[ x = [x_1, x_2] \]
  \[ x_1, x_2 \text{ conditionally independent given } f(x) \]
- Each half can be used to classify instance
  \[ f_1(x_1) \sim f_2(x_2) \sim f(x) \]
- Both \( f_1, f_2 \) are learnable
  \[ f_1 \in H_1, \quad f_2 \in H_2, \quad \exists \text{ learning algorithms } A_1, A_2 \]

Observations

- Can apply \( A_1 \) to generate as much training data as one wants
  - If \( x_1 \) is conditionally independent of \( x_2 / f(x) \),
  - then the error in the labels produced by \( A_1 \)
  - will look like random noise to \( A_2 \) !!!
- Thus no limit to quality of the hypothesis \( A_2 \) can make

Without Co-training

A Few Labeled Instances

- \( f_1(x_1) - f_2(x_2) \sim f(x) \)
- \( A_1 \) learns \( f_1 \) from \( x_1 \)
- \( A_2 \) learns \( f_2 \) from \( x_2 \)

Combine with ensemble?

Lots of Labeled Instances

Hypothesis

Unlabeled Instances

A1

A2

f1

f2

[\( x_1, x_2 \)]

f

Co-training

A Few Labeled Instances

- \( f_1(x_1) - f_2(x_2) \sim f(x) \)
- \( A_1 \) learns \( f_1 \) from \( x_1 \)
- \( A_2 \) learns \( f_2 \) from \( x_2 \)

Lots of Labeled Instances

Hypothesis

Unlabeled Instances

f1(x1)

f2

[\( x_1, x_2 \)]

f

Co-training Example

Prof. Domingos
Student Parag...
Projects: SRL,
Data mining
Teach a class on
Data mining
Jesse
Classes taken:
1. Data mining
2. Machine learning
Research: SRL

CSE 546: Data Mining
Course Description:
Topics:
Homework: ...
It really works!

- Learning to classify web pages as course pages
  - $x_1 =$ bag of words on a page
  - $x_2 =$ bag of words from all anchors pointing to a page

- Naive Bayes classifiers
  - 12 labeled pages
  - 1039 unlabeled

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Positive</th>
<th>Negative</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsupervised</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>Supervised</td>
<td>0.6</td>
<td>0.4</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Table: Error rate in percent for classifying web pages as course pages. The top row shows error when training on only the labeled examples. Bottom row shows errors when classifying using both labeled and unlabeled examples.