Introduction to Machine Learning

Rob Schapire
Princeton University

www.cs.princeton.edu/~schapire
Machine Learning

- studies how to **automatically learn** to make accurate predictions based on past observations
- **classification problems:**
  - classify examples into given set of categories

Diagram:
- labeled training examples
- machine learning algorithm
- classification rule
- new example
- predicted classification
Examples of Classification Problems

- **bioinformatics**
  - classify proteins according to their function
  - predict if patient will respond to particular drug/therapy based on microarray profiles
  - predict if molecular structure is a small-molecule binding site
- text categorization (e.g., spam filtering)
- fraud detection
- optical character recognition
- machine vision (e.g., face detection)
- natural-language processing (e.g., spoken language understanding)
- market segmentation (e.g.: predict if customer will respond to promotion)
Characteristics of Modern Machine Learning

- **primary goal**: highly accurate predictions on test data
  - goal is **not** to uncover underlying “truth”
- **methods should be** general purpose, fully automatic and “off-the-shelf”
  - however, in practice, incorporation of prior, human knowledge is crucial
- **rich interplay between** theory and practice
- emphasis on methods that can handle large datasets
Why Use Machine Learning?

**advantages:**
- often much more *accurate* than human-crafted rules (since data driven)
- humans often incapable of expressing what they know (e.g., rules of English, or how to recognize letters), but can easily classify examples
- automatic method to search for hypotheses explaining data
- cheap and flexible — can apply to any learning task

**disadvantages**
- need a lot of *labeled* data
- *error prone* — usually impossible to get perfect accuracy
- often difficult to discern what was learned
This Talk

- conditions for accurate learning
- two state-of-the-art algorithms:
  - boosting
  - support-vector machines
Conditions for Accurate Learning
Example: Good versus Evil

**problem**: identify people as good or bad from their appearance

<table>
<thead>
<tr>
<th></th>
<th>sex</th>
<th>mask</th>
<th>cape</th>
<th>tie</th>
<th>ears</th>
<th>smokes</th>
<th>class</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>training data</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>batman</td>
<td>male</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>Good</td>
</tr>
<tr>
<td>robin</td>
<td>male</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>Good</td>
</tr>
<tr>
<td>alfred</td>
<td>male</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>Good</td>
</tr>
<tr>
<td>penguin</td>
<td>male</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>Bad</td>
</tr>
<tr>
<td>catwoman</td>
<td>female</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>Bad</td>
</tr>
<tr>
<td>joker</td>
<td>male</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>Bad</td>
</tr>
<tr>
<td><strong>test data</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>batgirl</td>
<td>female</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>??</td>
</tr>
<tr>
<td>riddler</td>
<td>male</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>??</td>
</tr>
</tbody>
</table>
An Example Classifier

- **tie**
  - no
    - cape
      - no
        - bad
      - yes
        - good
  - yes
    - smokes
      - no
        - good
      - yes
        - bad
Another Possible Classifier

- perfectly classifies training data
- BUT: intuitively, overly complex
Yet Another Possible Classifier

- overly simple
- doesn’t even fit available data
**Learning the Central Problem of Machine Learning**

- Classifiers must be expressive enough to fit training data (so that "true" patterns are fully captured).
- **BUT:** Classifiers that are too complex may **overfit** (capture noise or spurious patterns in the data).
- **Problem:** Can’t tell best classifier complexity from training error.
- Controlling overfitting is the **central problem** of machine learning.
Building an Accurate Classifier

- for good test performance, need:
  - enough training examples
  - good performance on training set
  - classifier that is not too "complex" ("Occam's razor")
    - measure "complexity" by:
      - number of bits needed to write down
      - number of parameters
      - VC-dimension
- classifiers should be "as simple as possible, but no simpler"
- "simplicity" closely related to prior expectations
Theory

- can prove:

\[(\text{generalization error}) \leq (\text{training error}) + \tilde{O}\left(\sqrt{\frac{d}{m}}\right)\]

with high probability

- \(d = \text{VC-dimension}\)
- \(m = \text{number training examples}\)
**Example: Spam Filtering**

- **Problem**: filter out spam (junk email)

- **Gather large collection of examples of spam and non-spam**:
  
  ```
  From: yoav@att.com  Rob, can you review a paper...  non-spam 
  From: xa412@hotmail.com  Earn money without working!!!! ...  spam 
  ```

- **Main Observation**:
  - easy to find “rules of thumb” that are “often” correct
    - If ‘buy now’ occurs in message, then predict ‘spam’
  - hard to find single rule that is very highly accurate
The Boosting Approach

- devise computer program for deriving rough rules of thumb
- apply procedure to subset of emails
- obtain rule of thumb
- apply to 2nd subset of emails
- obtain 2nd rule of thumb
- repeat $T$ times
Details

- how to choose examples on each round?
  - concentrate on “hardest” examples (those most often misclassified by previous rules of thumb)
- how to combine rules of thumb into single prediction rule?
  - take (weighted) majority vote of rules of thumb

---

- can prove: if can always find weak rules of thumb slightly better than random guessing (51% accuracy), then can learn almost perfectly (99% accuracy) using boosting
AdaBoost

- given training examples
- initialize weights $D_1$ to be uniform across training examples
- for $t = 1, \ldots, T$:
  - train weak classifier ("rule of thumb") $h_t$ on $D_t$
  - compute new weights $D_{t+1}$:
    - decrease weight of examples correctly classified by $h_t$
    - increase weight of examples incorrectly classified by $h_t$
- output final classifier
  - $H_{\text{final}} = \text{weighted majority vote of } h_1, \ldots, h_T$
Toy Example

weak classifiers = vertical or horizontal half-planes
Round 1

\[ h_1 \]

\[ \varepsilon_1 = 0.30 \]
\[ \alpha_1 = 0.42 \]
Round 2

\[ \varepsilon_2 = 0.21 \]

\[ \alpha_2 = 0.65 \]
Round 3

\[ \varepsilon_3 = 0.14 \]
\[ \alpha_3 = 0.92 \]
Final Classifier

\[ H_{\text{final}} = \text{sign} \begin{pmatrix} 0.42 & +0.65 & +0.92 \\ + & + & + \end{pmatrix} \]
Theory of Boosting

- assume each weak classifier slightly better than random
- can prove training error drops to zero exponentially fast
- even so, naively expect significant overfitting, since a large number of rounds implies a large final classifier
- surprisingly, usually does not overfit
• test error does not increase, even after 1000 rounds
• test error continues to drop even after training error is zero!

**Explanation:**
- with more rounds of boosting, final classifier becomes more confident in its predictions
- increase in confidence implies better test error (regardless of number of rounds)
Support-Vector Machines
Geometry of SVM’s

- given **linearly separable** data
- **margin** = distance to separating hyperplane
- choose hyperplane that maximizes minimum margin
- intuitively:
  - want to separate +’s from –’s as much as possible
  - margin = measure of confidence
- support vectors = examples closest to hyperplane
Theoretical Justification

• let $\gamma = \text{minimum margin}$
  \[ R = \text{radius of enclosing sphere} \]

• then
  \[ \text{VC-dim} \leq \left( \frac{R}{\gamma} \right)^2 \]

  • so larger margins $\Rightarrow$ lower "complexity"
  • independent of number of dimensions

• in contrast, unconstrained hyperplanes in $\mathbb{R}^n$ have
  \[ \text{VC-dim} = (\# \text{ parameters}) = n + 1 \]
What If Not Linearly Separable?

- **answer #1:** penalize each point by distance must be moved to obtain large margin
- **answer #2:** map into higher dimensional space in which data becomes linearly separable
Example

- **not** linearly separable
- **map** $\mathbf{x} = (x_1, x_2) \mapsto \Phi(\mathbf{x}) = (1, x_1, x_2, x_1 x_2, x_1^2, x_2^2)$
- hyperplane in mapped space has form
  \[ a + bx_1 + cx_2 + dx_1 x_2 + ex_1^2 + f x_2^2 = 0 \]
  = conic in original space
- linearly separable in mapped space
Higher Dimensions Don’t (Necessarily) Hurt

- may project to very high dimensional space
- statistically, may not hurt since VC-dimension independent of number of dimensions \((R/\gamma)^2\)
- computationally, only need to be able to compute inner products
  \[
  \Phi(x) \cdot \Phi(z)
  \]
- sometimes can do very efficiently using kernels
Example (cont.)

• modify $\Phi$ slightly:

$$\Phi(x) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2)$$

• then

$$\Phi(x) \cdot \Phi(z) = 1 + 2x_1z_1 + 2x_2z_2 + 2x_1x_2z_1z_2 + x_1^2z_1^2 + x_2^2 + z_2^2$$

$$= (1 + x_1z_1 + x_2z_2)^2$$

$$= (1 + x \cdot z)^2$$

• in general, for polynomial of degree $d$, use $(1 + x \cdot z)^d$

• very efficient, even though finding hyperplane in $O(n^d)$ dimensions
Kernels

- kernel = function $K$ for computing
  \[ K(x, z) = \Phi(x) \cdot \Phi(z) \]

- permits efficient computation of SVM’s in very high dimensions

- many kernels have been proposed and studied
  - provides power, versatility and opportunity for incorporation of prior knowledge
Significance of SVM’s and Boosting

- grounded in rich theory with provable guarantees
- flexible and general purpose
- off-the-shelf and fully automatic
- fast and easy to use
- able to work effectively in very high dimensional spaces
- performs well empirically in many experiments and in many applications
Summary

- central issues in machine learning:
  - avoidance of overfitting
  - balance between simplicity and fit to data
- quick look at two learning algorithms: boosting and SVM’s
- many other algorithms not covered:
  - decision trees
  - neural networks
  - nearest neighbor algorithms
  - Naive Bayes
  - bagging
- also, classification just one of many problems studied in machine learning
Other Machine Learning Problem Areas

- **supervised learning**
  - classification
  - regression – predict real-valued labels
  - rare class / cost-sensitive learning
- **unsupervised** – no labels
  - clustering
  - density estimation
- **semi-supervised**
  - in practice, unlabeled examples much cheaper than labeled examples
  - how to take advantage of both labeled and unlabeled examples
  - active learning – how to carefully select which unlabeled examples to have labeled
Further reading on machine learning in general:


Boosting:


Many more papers, tutorials, etc. available at www.boosting.org.

Support-vector machines:


Many more papers, tutorials, etc. available at www.kernel-machines.org.