Announcements

- On-going: contest (optional and FUN!)
- Remaining lectures:
  - Today: Machine Learning: Nearest Neighbors, Kernels
  - Wednesday: Machine Learning for Computer Vision
  - Next Monday: Case Studies in Speech/Language and Robotics
  - Next Wednesday:
    - Course Wrap-Up
    - Pointers to courses and Books for those who want to learn more AI
    - Contest!
    - RRR Week Monday and Wednesday: Review Sessions

Today

- Nearest neighbors
- Kernels
- Applications:
  - Extension to ranking / web-search
  - Pacman apprenticeship

Classification overview

- Naive Bayes:
  - Builds a model/training data
  - Gives prediction probabilities
  - Strong assumptions about feature independence
  - One pass through data (counting)
- Perceptron:
  - Makes less assumptions about data
  - Mistake-driven learning
  - Multiple passes through data (prediction)
  - Often more accurate
- MIRA:
  - Like perceptron, but adaptive scaling of size of update
- SVM:
  - Properties similar to perceptron
  - Convex optimization formulation
- Nearest-Neighbor:
  - Non-parametric: more expressive with more training data
- Kernels
  - Efficient way to make linear learning architectures into nonlinear ones

Case-Based Reasoning

- Similarity for classification
  - Case-based reasoning
  - Predict an instance’s label using similar instances
- Nearest-neighbor classification
  - 1-NN: copy the label of the most similar data point
  - K-NN: let the k nearest neighbors vote (have to devise a weighting scheme)
  - Key issue: how to define similarity
  - Trade-off:
    - Small k gives relevant neighbors
    - Large k gives smoother functions
    - Sound familiar?

[Demo]
Parametric / Non-parametric

- **Parametric models:**
  - Fixed set of parameters
  - More data means better settings

- **Non-parametric models:**
  - Complexity of the classifier increases with data
  - Better in the limit, often worse in the non-limit

(K)NN is **non-parametric**

<table>
<thead>
<tr>
<th>2 Examples</th>
<th>10 Examples</th>
<th>100 Examples</th>
<th>100000 Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="2 Examples" /></td>
<td><img src="image2.png" alt="10 Examples" /></td>
<td><img src="image3.png" alt="100 Examples" /></td>
<td><img src="image4.png" alt="100000 Examples" /></td>
</tr>
</tbody>
</table>

Nearest-Neighbor Classification

- Nearest neighbor for digits:
  - Take new image
  - Compare to all training images
  - Assign based on closest example

- Encoding: image is vector of intensities:
  \[ \mathbf{1} = (0.0, 0.0, 0.3, 0.6, 0.7, 0.1, \ldots, 0.0) \]

- What’s the similarity function?
  - Dot product of two images vectors:
    \[ \text{sim}(\mathbf{x}, \mathbf{x'}) = \mathbf{x} \cdot \mathbf{x'} = \sum_i x_i x'_i \]
  - Usually normalize vectors so \( ||\mathbf{x}|| = 1 \)
  - \( \min = 0 \), \( \max = 1 \)

Basic Similarity

- Many similarities based on feature dot products:
  \[ \text{sim}(\mathbf{x}, \mathbf{x'}) = f(x) \cdot f(x') = \sum_i f_i(x) f_i(x') \]

- If features are just the pixels:
  \[ \text{sim}(\mathbf{x}, \mathbf{x'}) = \mathbf{x} \cdot \mathbf{x'} = \sum_i x_i x'_i \]

- Note: not all similarities are of this form

Invariant Metrics

- Better distances use knowledge about vision
- Invariant metrics:
  - Similarities are invariant under certain transformations
  - Rotation, scaling, translation, stroke-thickness…
  - E.g.:
    \[ \text{E.g.:} \]
    \[ 16 \times 16 = 256 \text{ pixels; a point in 256-dim space} \]
    \[ \text{Small similarity in } \mathbb{R}^{256} \text{ (why?)} \]
    \[ \text{Variety of invariant metrics in literature} \]
    \[ \text{Viable alternative: transform training examples such that training set includes all variations} \]

Rotation Invariant Metrics

- Each example is now a curve in \( \mathbb{R}^{256} \)
- Rotation invariant similarity:
  \[ s'(r) = \max_s ||r|| \]
  - E.g. highest similarity between images’ rotation lines

Classification overview

- Naive Bayes
- Perceptron, MIRA
- SVM
- Nearest-Neighbor
- Kernels
A Tale of Two Approaches …

- Nearest neighbor-like approaches
  - Can use fancy similarity functions
  - Don’t actually get to do explicit learning

- Perceptron-like approaches
  - Explicit training to reduce empirical error
  - Can’t use fancy similarity, only linear
  - Or can they? Let’s find out!

Perceptron Weights

- What is the final value of a weight \( w \), of a perceptron?
  - Can it be any real vector?
  - No! It’s built by adding up inputs.

\[
\begin{align*}
\omega_y &= 0 + f(x_1) - f(x_2) + \ldots \\
        &= \sum_{i} \alpha_{i,y} f(x_i)
\end{align*}
\]

- Can reconstruct weight vectors (the primal representation) from update counts (the dual representation)

\[
\alpha_y = \langle \alpha_{1,y}, \alpha_{2,y}, \ldots, \alpha_{n,y} \rangle
\]

Dual Perceptron

- How to classify a new example \( x \)?

\[
\text{score}(y, x) = \omega_y \cdot f(x) = \sum_{i} \alpha_{i,y} K(x_i, x)
\]

- If someone tells us the value of \( K \) for each pair of examples, never need to build the weight vectors!

Dual Perceptron

- Start with zero counts (alpha)
- Pick up training instances one by one
- Try to classify \( x_n \)
  - If correct, no change!
  - If wrong: lower count of wrong class (for this instance), raise score of right class (for this instance)

\[
\begin{align*}
\alpha_{y,n} &= \alpha_{y,n} - 1 \quad \Rightarrow \quad \omega_y = \omega_y - f(x_n) \\
\alpha_{y^*,n} &= \alpha_{y^*,n} + 1 \quad \Rightarrow \quad \omega_{y^*} = \omega_{y^*} + f(x_n)
\end{align*}
\]

Kernelized Perceptron

- If we had a black box (kernel) which told us the dot product of two examples \( x \) and \( y \):
  - Could work entirely with the dual representation
  - No need to ever take dot products (“kernel trick”)  

\[
\text{score}(y, x) = \omega_y \cdot f(x) = \sum_{i} \alpha_{i,y} K(x_i, x)
\]

- Like nearest neighbor – work with black-box similarities
- Downside: slow if many examples get nonzero alpha

Kernelized MIRA

- Our formula for \( \tau \) (see last lecture)

\[
\begin{align*}
\alpha_{y,n} &= \alpha_{y,n} - \tau \\
\alpha_{y^*,n} &= \alpha_{y^*,n} + \tau \\
\omega_y &= \omega_y - \tau f(x) \\
\omega_{y^*} &= \omega_{y^*} + \tau f(x)
\end{align*}
\]

\[
\tau^* = \min \left( \sum_{i} \alpha_{i,y} K(x_i, x) - \sum_{i} \alpha_{i,y^*} K(x_i, x) + 1, 0 \right)_{\text{prime}}
\]

\[
\tau^* = \min \left( \sum_{i} \alpha_{i,y} K(x_i, x) - \sum_{i} \alpha_{i,y^*} K(x_i, x) + 1, 0 \right)_{\text{dual}}
\]
Kernels: Who Cares?

- So far: a very strange way of doing a very simple calculation
- "Kernel trick": we can substitute any* similarity function in place of the dot product
- Lets us learn new kinds of hypothesis

* Fine print: if your kernel doesn’t satisfy certain technical requirements, lots of proofs break. E.g. convergence, mistake bounds. In practice, illegal kernels sometimes work (but not always).

Non-Linear Separators

- Data that is linearly separable (with some noise) works out great:
- But what are we going to do if the dataset is just too hard?

Some Kernels

- Kernels implicitly map original vectors to higher dimensional spaces, take the dot product there, and hand the result back
- Linear kernel: \( K(x, x') = x \cdot x' = \sum_i x_i x'_i \)
- Quadratic kernel: \( K(x, x') = (x \cdot x' + 1)^2 \)

For \( \phi \in \mathbb{R}^3 \):

\[ \phi(x) = [x_1, x_2, x_3, \sqrt{dx_1^{-1} x_2} \sqrt{dx_1^{-1} x_3} \ldots \sqrt{dx_1} \sqrt{dx_2} \sqrt{dx_3}] \]

For \( \phi \in \mathbb{R}^d \) the \( d \)-order polynomial kernel’s implicit feature space is \( \binom{d+1}{2} \) dimensional.

By contrast, computing the kernel directly only requires \( O(d) \) time.
Some Kernels (3)

- Kernels implicitly map original vectors to higher dimensional spaces, take the dot product there, and hand the result back.
- Radial Basis Function (or Gaussian) Kernel: infinite dimensional representation
  \[ K(x, x') = \exp\left(-\frac{||x - x'||^2}{2\sigma^2}\right) \]
- Discrete kernels: e.g. string kernels
  - Features: all possible strings up to some length
  - To compute kernel: don’t need to enumerate all substrings for each word, but only need to find strings appearing in both \( x \) and \( x' \).

Why Kernels?

- Can’t you just add these features on your own (e.g. add all pairs of features instead of using the quadratic kernel)?
  - Yes, in principle, just compute them
  - No need to modify any algorithms
  - But, number of features can get large (or infinite)
- Kernels let us compute with these features implicitly
  - Example: implicit dot product in polynomial, Gaussian and string kernel takes much less space and time per dot product
  - Of course, there’s the cost for using the pure dual algorithms: you need to compute the similarity to every training datum.

Recap: Classification

- Classification systems:
  - Supervised learning
  - Make a prediction given evidence
  - We’ve seen several methods for this
  - Useful when you have labeled data

Extension: Web Search

- Information retrieval:
  - Given information needs, produce information
  - Includes, e.g. web search, question answering, and classic IR
- Web search: not exactly classification, but rather ranking

Feature-Based Ranking

- \( x = \text{"Apple Computers"} \)
  - \( f(x) = [0.3 \ 5 \ 0 \ 0 \ldots] \)
  - \( f(x) = [0.8 \ 4 \ 2 \ 1 \ldots] \)

Perceptron for Ranking

- Inputs \( x \)
- Candidates \( y \)
- Many feature vectors: \( f(x, y) \)
- One weight vector: \( w \)
  - Prediction:
    \[ y = \text{arg max}_y \ w \cdot f(x, y) \]
  - Update (if wrong):
    \[ w = w + f(x, y^*) - f(x, y) \]

Pacman Apprenticeship!

- Examples are states $s$
- Candidates are pairs $(s,a)$
- "Correct" actions: those taken by expert
- Features defined over $(s,a)$ pairs: $f(s,a)$
- Score of a q-state $(s,a)$ given by:
  $$ w \cdot f(s,a) $$
- How is this VERY different from reinforcement learning?