CSE 40171: Artificial Intelligence

Learning Algorithms: Kernel Machines
Homework #7 has been released.
It is due 11/16 at 11:59PM.
Group Project Updates are due on 11/12 at 11:59PM
Quiz #2 will be given on 11/14 in class. See topic checklist posted to the course website.

Review will happen on Monday
What do we do if our data are not linearly separable?
Case-Based Learning
Non-Separable Data
Case-Based Reasoning

Classification from similarity

- Case-based reasoning
- Predict an instance’s label using similar instances
Case-Based Reasoning

Nearest-neighbor classification

- 1-NN: copy the label of the most similar data point
- K-NN: vote the $k$ nearest neighbors (need a weighting scheme)
- Key issue: how to define similarity
- Trade-offs: Small $k$ gives relevant neighbors, Large $k$ gives smoother functions
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

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188
Nearest-Neighbor Classification

Nearest neighbor for digits:

- Take new image
- Compare to all training images
- Assign based on closest example

\[ \mathbf{1} = \langle 0.0 \ 0.0 \ 0.3 \ 0.8 \ 0.7 \ 0.1 \ldots 0.0 \rangle \]

Encoding: image is vector of intensities:

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188
Nearest-Neighbor Classification

What’s the similarity function?

- Dot product of two images vectors?

\[ \text{sim}(x, x') = x \cdot x' = \sum_i x_i x'_i \]

- Usually normalize vectors so \( ||x|| = 1 \)
- \( \min = 0 \) (when?), \( \max = 1 \) (when?)
Similarity Functions
Basic Similarity

Many similarities based on **feature dot products**:

\[
sim(x, x') = f(x) \cdot f(x') = \sum_i f_i(x) f_i(x')
\]

If features are just the pixels:

\[
sim(x, x') = x \cdot x' = \sum_i x_i x_i'
\]

Note: not all similarities are of this form
Invariant Metrics

Better similarity functions use knowledge about vision

Example: invariant metrics:

- Similarities are invariant under certain transformations
- Rotation, scaling, translation, stroke-thickness…

Example:

- 16 x 16 = 256 pixels; a point in 256-dim space
- These points have small similarity in $\mathbb{R}^{256}$ (why?)

How can we incorporate such invariances into our similarities?
Rotation Invariant Metrics

Each example is now a curve in $\mathbb{R}^{256}$

Rotation invariant similarity:

$$s' = \max s( r(3), r(3))$$

e.g., highest similarity between images’ rotation lines
Template Deformation

Deformable templates:

- An “ideal” version of each category
- Best-fit to image using min variance
- Cost for high distortion of template
- Cost for image points being far from distorted template

Used in many commercial digit recognizers

Examples from [Hastie 94]
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!
Kernelization
Perceptron Weights

What is the final value of a weight $w_y$ of a perceptron?

- Can it be any real vector?
- No! It’s built by adding up inputs.

$$w_y = 0 + f(x_1) - f(x_5) + \ldots$$

$$w_y = \sum_{i} \alpha_{i,y} f(x_i)$$

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 $x$?

\[ \text{score}(y, x) = w_y \cdot f(x) \]

\[ = \left( \sum_i \alpha_{i,y} f(x_i) \right) \cdot f(x) \]

\[ = \sum_i \alpha_{i,y} (f(x_i) \cdot f(x)) \]

\[ = \sum_i \alpha_{i,y} K(x_i, x) \]

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

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188
Dual Perceptron

Start with zero counts (alpha)
Pick up training instances one by one
Try to classify $x_n$

$$y = \arg \max_y \sum_i \alpha_{i,y} K(x_i, x_n)$$

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188
Kernelized Perceptron

If correct, no change!

If wrong: lower count of wrong class (for this instance), raise count of right class (for this instance)

\[ \alpha_{y,n} = \alpha_{y,n} - 1 \]
\[ \alpha_{y^*,n} = \alpha_{y^*,n} + 1 \]

\[ w_y = w_y - f(x_n) \]
\[ w_{y^*} = w_{y^*} + f(x_n) \]
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 hypotheses

* 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-Linearity
Non-Linear Separators

Data that is linearly separable works out great for linear decision rules:

But what are we going to do if the dataset is just too hard?

How about… mapping data to a higher-dimensional space:
Non-Linear Separators

General idea: the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable:
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: \( K(x, x') = (x \cdot x' + 1)^2 \)

RBF: infinite dimensional representation \( \sum_{i,j} x_i x_j x_i' x_j' + 2 \sum_i x_i x_i' + 1 \)

Discrete kernels: e.g., string kernels \( K(x, x') = \exp(-||x - x'||^2) \)

Slide credit: Ray Mooney, UT
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)
- Some kernels not as usefully thought of in their expanded representation, e.g., RBF kernels
Why Kernels?

Kernels let us compute with these features implicitly

- Example: implicit dot product in quadratic 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