CS 188: Artificial Intelligence
Fall 2007

Lecture 25: Kernels

11/27/2007

Dan Klein – UC Berkeley

Feature Extractors

- A feature extractor maps inputs to feature vectors

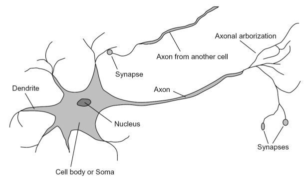
```
Dear Sir.

First, I must
solicit your
confidence in
this
transaction,
this is by
virtue of its
nature as being
utterly
confidencial and
top secret. …
```

```
W=dear      :  1
W=sir       :  1
W=this      :  2
...
W=wish      :  0
...
MISSPELLED :  2
NAMELESS    :  1
ALL_CAPS    :  0
NUM_URLS    :  0
...
```

- Many classifiers take feature vectors as inputs
- Feature vectors usually very sparse, use sparse encodings (i.e. only represent non-zero keys)

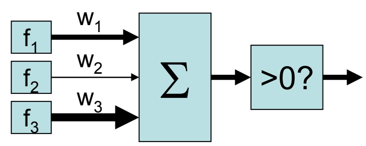
The Binary Perceptron

- Inputs are features
- Each feature has a weight
- Sum is the activation



$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x)$$

- If the activation is:
  - Positive, output 1
  - Negative, output 0



Example: Spam

- Imagine 4 features:
  - Free (number of occurrences of "free")
  - Money (occurrences of "money")
  - BIAS (always has value 1)

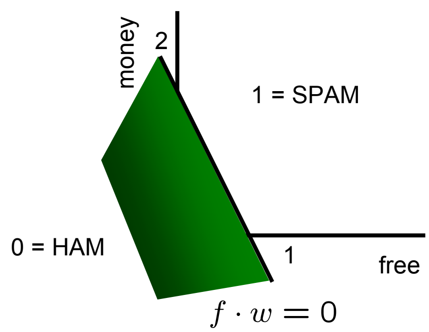
x	$f(x)$	w	$\sum_i w_i \cdot f_i(x)$
"free money"	BIAS : 1	BIAS : -3	$(1)(-3)$ +
	free : 1	free : 4	$(1)(4)$ +
	money : 1	money : 2	$(1)(2)$ +
	the : 0	the : 0	$(0)(0)$ +
	...	...	...
			$= 3$

Binary Decision Rule

- In the space of feature vectors
  - Any weight vector is a hyperplane
  - One side will be class 1
  - Other will be class 0

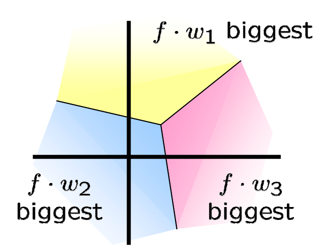
w

BIAS	:	-3
free	:	4
money	:	2
the	:	0
...		



The Multiclass Perceptron

- If we have more than two classes:
  - Have a weight vector for each class
  - Calculate an activation for each class



$$\text{activation}_w(x, c) = \sum_i w_{c,i} \cdot f_i(x)$$

- Highest activation wins

$$c = \arg\max_c \left(\text{activation}_w(x, c)\right)$$

Example

"win the vote" →

BIAS	:	1
win	:	1
game	:	0
vote	:	1
the	:	1
...		

w_{SPORTS}

BIAS	:	-2
win	:	4
game	:	4
vote	:	0
the	:	0
...		

$w_{POLITICS}$

BIAS	:	1
win	:	2
game	:	0
vote	:	4
the	:	0
...		

w_{TECH}

BIAS	:	2
win	:	0
game	:	2
vote	:	0
the	:	0
...		

The Perceptron Update Rule

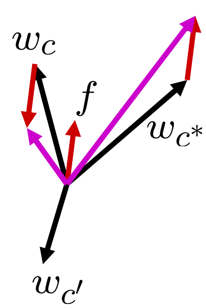
- Start with zero weights
- Pick up training instances one by one
- Try to classify

$$c = \arg\max_c \; w_c \cdot f(x)$$
$$= \arg\max_c \; \textstyle\sum_i w_{c,i} \cdot f_i(x)$$

- If correct, no change!
- If wrong: lower score of wrong answer, raise score of right answer

$$w_c = w_c - f(x)$$

$$w_{c^*} = w_{c^*} + f(x)$$



Example

"win the vote"

"win the election"

"win the game"

w_{SPORTS}

```
BIAS  :
win   :
game  :
vote  :
the   :
...
```

$w_{POLITICS}$

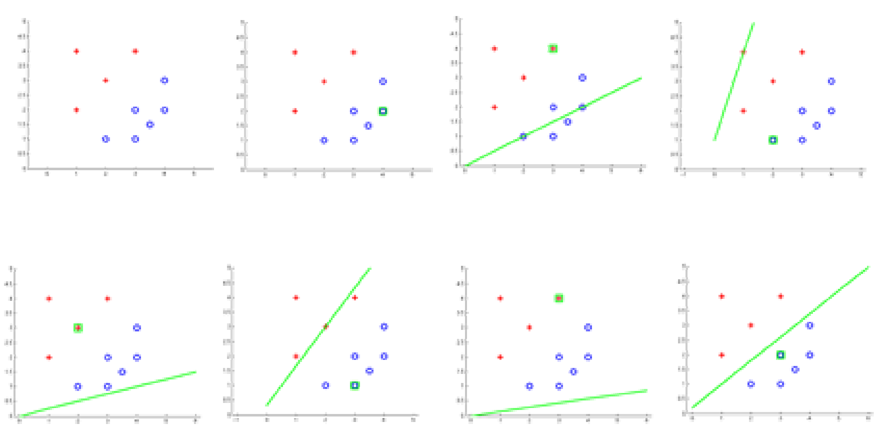
```
BIAS  :
win   :
game  :
vote  :
the   :
...
```

w_{TECH}

```
BIAS  :
win   :
game  :
vote  :
the   :
...
```

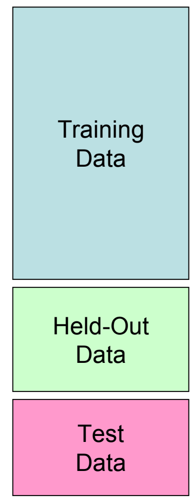
Examples: Perceptron

- Separable Case



Mistake-Driven Classification

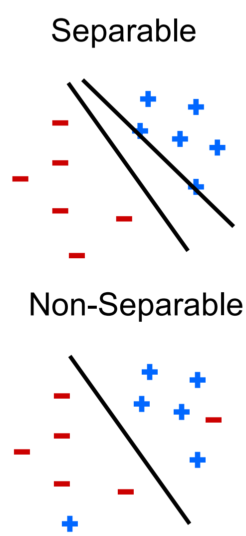
- In naïve Bayes, parameters:
  - From data statistics
  - Have a causal interpretation
  - One pass through the data
- For the perceptron parameters:
  - From reactions to mistakes
  - Have a discriminative interpretation
  - Go through the data until held-out accuracy maxes out



Properties of Perceptrons

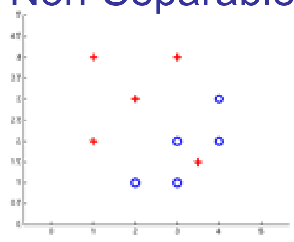
- Separability: some parameters get the training set perfectly correct
- Convergence: if the training is separable, perceptron will eventually converge (binary case)
- Mistake Bound: the maximum number of mistakes (binary case) related to the *margin* or degree of separability

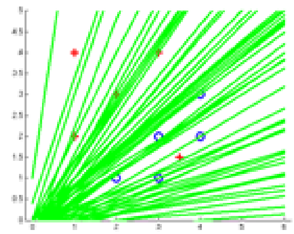
$$\text{mistakes} < \frac{1}{\delta^2}$$

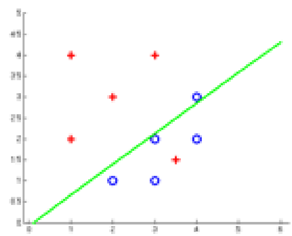


Examples: Perceptron

- Non-Separable Case

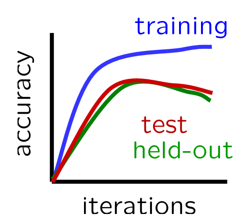




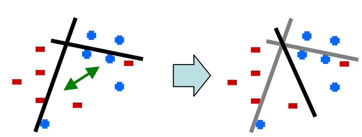


Issues with Perceptrons

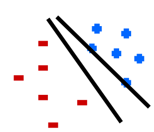
- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining isn't quite as bad as overfitting, but is similar



- Regularization: if the data isn't separable, weights might thrash around
  - Averaging weight vectors over time can help (averaged perceptron)

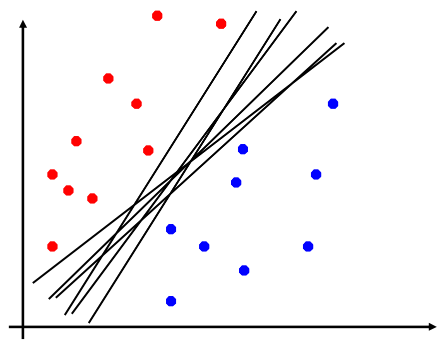


- Mediocre generalization: finds a "barely" separating solution



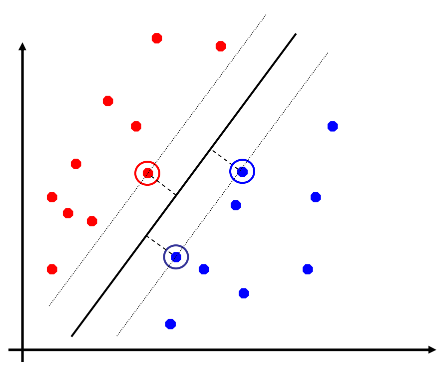
Linear Separators

- Which of these linear separators is optimal?



Support Vector Machines

- Maximizing the margin: good according to intuition and PAC theory.
- Only support vectors matter; other training examples are ignorable.
- Support vector machines (SVMs) find the separator with max margin
- Mathematically, gives a quadratic program to solve
- Basically, SVMs are perceptrons with smarter update counts!



Summary

- Naïve Bayes
  - Build classifiers using model of training data
  - Smoothing estimates is important in real systems
  - Classifier confidences are useful, when you can get them

- Perceptrons:
  - Make less assumptions about data
  - Mistake-driven learning
  - Multiple passes through data

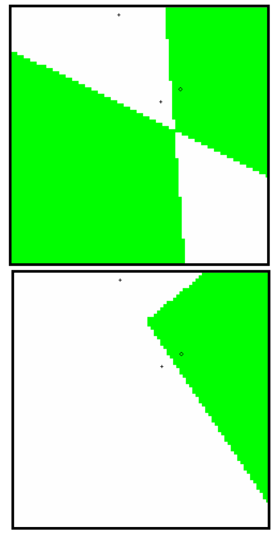
Similarity Functions

- Similarity functions are very important in machine learning

- Topic for next class: kernels
  - Similarity functions with special properties
  - The basis for a lot of advance machine learning (e.g. SVMs)

Case-Based Reasoning

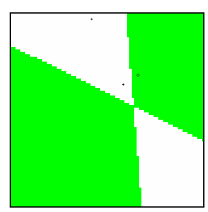
- Similarity for classification
  - Case-based reasoning
  - Predict an instance's label using similar instances
- Nearest-neighbor classification
  - 1-NN: copy the label of the most similar data point
  - K-NN: let the k nearest neighbors vote (have to devise a weighting scheme)
  - Key issue: how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



http://www.cs.cmu.edu/~zhuxj/courseproject/knndemo/KNN.html

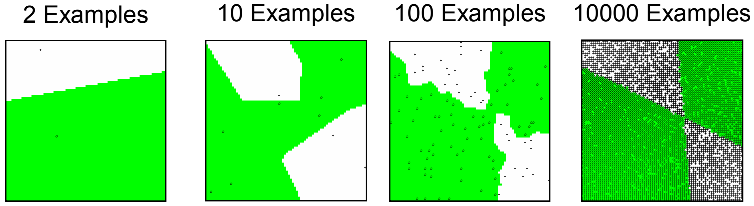
Parametric / Non-parametric

- 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



Truth

2 Examples | 10 Examples | 100 Examples | 10000 Examples



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:

 $= \langle 0.0\ \ 0.0\ \ 0.3\ \ 0.8\ \ 0.7\ \ 0.1 \ldots 0.0 \rangle$

- What's the similarity function?
  - Dot product of two images vectors?

$$\text{sim}(x, y) = x \cdot y = \sum_i x_i y_i$$

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

Basic Similarity

- Many similarities based on feature dot products:

$$\text{sim}(x, y) = f(x) \cdot f(y) = \sum_i f_i(x) f_i(y)$$

- If features are just the pixels:

$$\text{sim}(x, y) = x \cdot y = \sum_i x_i y_i$$

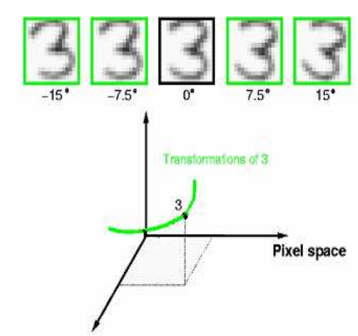
- Note: not all similarities are of this form

Invariant Metrics

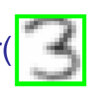
- Better distances use knowledge about vision
- Invariant metrics:
  - Similarities are invariant under certain transformations
  - Rotation, scaling, translation, stroke-thickness…
  - E.g:  
    - 16 x 16 = 256 pixels; a point in 256-dim space
    - Small similarity in R^{256} (why?)
  - How to incorporate invariance into similarities?

This and next few slides adapted from Xiao Hu, UIUC

Rotation Invariant Metrics

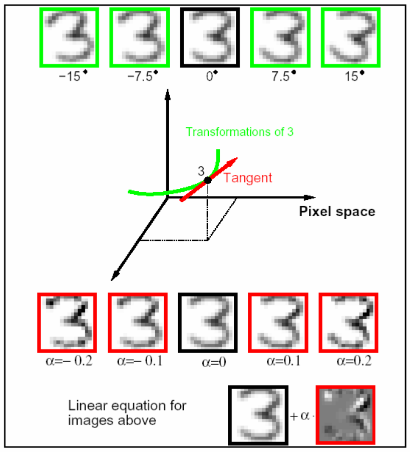


- Each example is now a curve in R^{256}
- Rotation invariant similarity:

  s'=max s(r(), r())

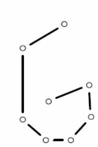
- E.g. highest similarity between images' rotation lines

Tangent Families

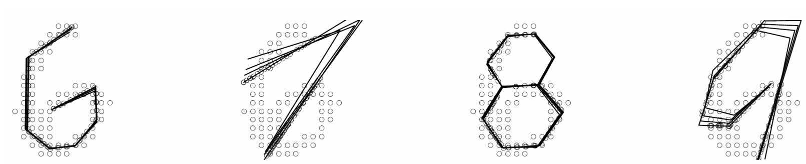


- Problems with s':
  - Hard to compute
  - Allows large transformations (6 → 9)
- Tangent distance:
  - 1st order approximation at original points.
    - Easy to compute
    - Models small rotations

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 kernels (similarity functions)
  - Don't actually get to do explicit learning
- Perceptron-like approaches
  - Explicit training to reduce empirical error
  - Can't use fancy kernels (why not?)
  - Or can you? Let's find out!

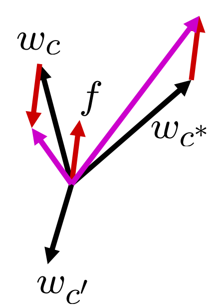
The Perceptron, Again

- Start with zero weights
- Pick up training instances one by one
- Try to classify

$$c = \arg\max_c \; w_c \cdot f(x)$$
$$= \arg\max_c \; \textstyle\sum_i w_{c,i} \cdot f_i(x)$$

- If correct, no change!
- If wrong: lower score of wrong answer, raise score of right answer

$$w_c = w_c - f(x)$$
$$w_{c^*} = w_{c^*} + f(x)$$



Perceptron Weights

- What is the final value of a weight w_c?
  - Can it be any real vector?
  - No! It's built by adding up inputs.

$$w_c = 0 + f(x_1) - f(x_5) + \dots$$

$$w_c = \sum_i \alpha_{i,c} f(x_i)$$

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

$$\alpha_c = \langle \alpha_{1,c} \;\; \alpha_{2,c} \;\; \dots \;\; \alpha_{n,c} \rangle$$

Dual Perceptron

- How to classify a new example x?

$$\begin{aligned} \text{score}(c, x) &= w_c \cdot f(x) \\ &= \left(\sum_i \alpha_{i,c} f(x_i) \right) \cdot f(x) \\ &= \sum_i \alpha_{i,c} (f(x_i) \cdot f(x)) \\ &= \sum_i \alpha_{i,c} \, K(x_i, x) \end{aligned}$$

- 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,

$$c = \arg\max_c \sum_i \alpha_{i,c} K(x_i, x)$$

- If correct, no change!
- If wrong: lower count of wrong class (for this instance), raise score of right class (for this instance)

$$\alpha_{c,n} = \alpha_{c,n} - 1 \qquad w_c = w_c - f(x)$$

$$\alpha_{c^*,n} = \alpha_{c^*,n} + 1 \qquad w_{c^*} = w_{c^*} + f(x)$$

Kernelized Perceptron

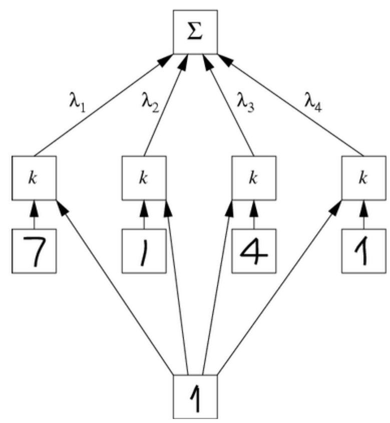
- If we had a black box (kernel) which told us the dot product of two examples x and y:
  - Could work entirely with the dual representation
  - No need to ever take dot products ("kernel trick")

$$\text{score}(c, x) = w_c \cdot f(x)$$

$$= \sum_i \alpha_{i,c} K(x_i, x)$$

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

Kernelized Perceptron Structure



$$\sum = \text{score}(c, x)$$

$$\lambda_i = \alpha_{c,i}$$

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

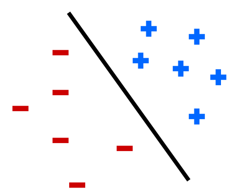
* Fine print: if your kernel doesn't satisfy certain technical requirements, lots of proofs break. E.g. convergence, mistake bounds. In practice, illegal kernels *sometimes* work (but not always).

Properties of Perceptrons

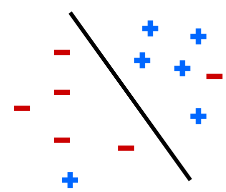
- Separability: some parameters get the training set perfectly correct
- Convergence: if the training is separable, perceptron will eventually converge (binary case)
- Mistake Bound: the maximum number of mistakes (binary case) related to the *margin* or degree of separability

$$\text{mistakes} < \frac{1}{\delta^2}$$

Separable

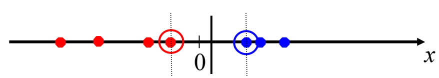


Non-Separable

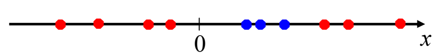


Non-Linear Separators

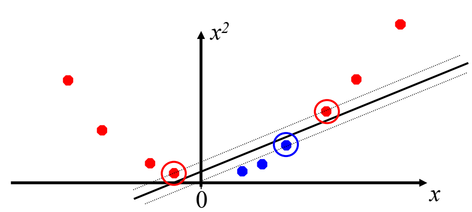
- Data that is linearly separable (with some noise) works out great:



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



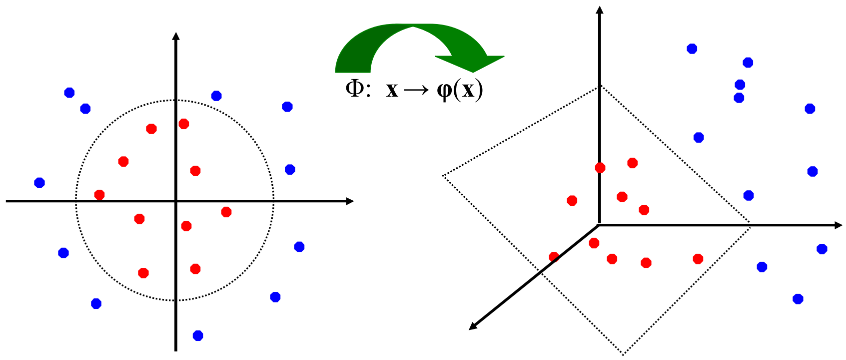
- How about… mapping data to a higher-dimensional space:



This and next few slides adapted from Ray Mooney, UT

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: $K(x, x') = (x \cdot x' + 1)^2$

$$= \sum_{i,j} x_i x_j x'_i x'_j + 2 \sum_i x_i x'_i + 1$$

- RBF: infinite dimensional representation

$$K(x, x') = \exp(-||x - x'||^2)$$

- Discrete kernels: e.g. string kernels