Mistake-Driven Classification

- The multi-class perceptron has a weight vector $w_y$ for each label value $y$
- The label is predicted by:
  \[ y = \arg \max_y w_y \cdot f(x) \]
- Predicts the label with highest activation
- Weights are updated whenever the classifier makes a mistake on a training datum (whenever predicted $y$ is not $y^*$)

The Perceptron Update Rule

- Start with zero weights
- Pick up training instances one by one
- Classify with current weights
  \[ y = \arg \max_y w_y \cdot f(x) \]
- If correct, no change!
- If wrong: lower score of wrong answer, raise score of right answer
  \[ w_y = w_y - f(x) \]
  \[ w_{y^*} = w_{y^*} + f(x) \]

Fixing the Perceptron

- Idea: adjust the size of the weight update to mitigate these effects
- MIRA*: choose an update size that fixes the current mistake…
  \[ \min_w \frac{1}{2} \sum_y ||w_y - w_y'||^2 \]
- … but, minimizes the change to $w$
- Assesses separable data
- Real data is never separable

* Margin Infused Relaxed Algorithm
Minimum Correcting Update

\[ \min_w \frac{1}{2} \sum_y ||w_y - w'_y||^2 \]

\[ w_y \cdot f \geq w_y \cdot f + 1 \]

\[ \min ||f||^2 \]

\[ w_y \cdot f \geq w_y \cdot f + 1 \]

\[ (w'_y + \tau f) \cdot f = (w'_y - \tau f) \cdot f + 1 \]

\[ \tau = \frac{(w'_y - \tau f) \cdot f + 1}{2f \cdot f} \]

\[ \tau = 0 \]

min not \( \tau \neq 0 \), or would not have made an error, so min will be where equality holds

MIRA

- In practice, it's bad to make updates that are too large
  - Example may be labeled incorrectly
  - Solution: cap the maximum possible value of \( \tau \) with some constant \( C \)
  - Corresponds to an optimization that assumes non-separable data
  - Usually converges faster than perceptron
  - Usually performs better, especially on noisy data

Linear Separators

- Which of these linear separators is optimal?

Support Vector Machines

- Maximizing the margin: good according to intuition and theory.
  - Only support vectors matter; other training examples are ignorable.
  - Support vector machines (SVMs) find the separator with max margin
  - Basically, SVMs are MIRA where you optimize over all examples at once

Summary

- Naïve Bayes
  - Build classifiers using model of training data
  - Smoothing estimates is important in real systems
  - One pass through data

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

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
  - 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

**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:
  \[ \mathbf{1} = (0.0 \ 0.0 \ 0.3 \ 0.8 \ 0.7 \ 0.1 \ldots 0.0) \]
- 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 \( ||x|| = 1 \)
  - \( \text{min} = 0 \) (when?), \( \text{max} = 1 \) (when?)

**Basic Similarity**

- Many similarities based on feature dot products:
  \[ \text{sim}(x, x') = f(x) \cdot f(x') = \sum_i f_i(x) f_i(x') \]
- If features are just the pixels:
  \[ \text{sim}(x, x') = x \cdot x' = \sum_i x_i x'_i \]
- 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:
    - 16 x 16 = 256 pixels; a point in 256-dim space
    - Small similarity in \( \mathbb{R}^{256} \) (why?)
  - How to incorporate invariance into similarities?

**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

**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 \( w_i \) 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)

\[
\alpha_y = (\alpha_{1,y}, \alpha_{2,y}, \ldots, \alpha_{n,y})
\]

### 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!

### Dual Perceptron

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

\[
y = \arg \max_y \sum_i \alpha_{i,y} K(x_i, x)
\]
- If correct, no change!
- If wrong: lower count of wrong class (for this instance), raise score of right class (for this instance)

\[
\alpha_{y,n} = \alpha_{y,n} - 1
\]
\[
w_y = w_y - f(x)
\]
\[
\alpha_{y^*,n} = \alpha_{y^*,n} + 1
\]
\[
w_{y^*} = w_{y^*} + f(x)
\]

### 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")

\[
\text{score}(y, x) = w_y \cdot f(x) = \sum_i \alpha_{i,y} K(x_i, x)
\]
- Like nearest neighbor – work with black-box similarities
- Downside: slow if many examples get nonzero alpha

### 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?

- 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:

\[ \Phi: x \rightarrow \phi(x) \]

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)
  - Some kernels not as usefully thought of in their expanded representation, e.g. RBF or data-defined kernels [Henderson and Titov 05]

- 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