Lecture 16: Machine Learning Topics
12/7/2010

Luke Zettlemoyer

Most slides over the course adapted from Dan Klein.
Announcements

- Syllabus revised
  - Machine learning focus
- We will do mini-project status reports during last class, on Thursday
  - Instructions were emailed and are on web page
Outline

- Learning: Naive Bayes and Perceptron
  - (Recap) Perceptron
  - MIRA
  - SVMs
  - Linear Ranking Models
  - Nearest neighbor
  - Kernels
  - Clustering
Generative vs. Discriminative

- **Generative classifiers:**
  - E.g. naïve Bayes
  - A joint probability model with evidence variables
  - Query model for causes given evidence

- **Discriminative classifiers:**
  - No generative model, no Bayes rule, often no probabilities at all!
  - Try to predict the label Y directly from X
  - Robust, accurate with varied features
  - Loosely: mistake driven rather than model driven
(Recap) Linear Classifiers

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

\[ \text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x) \]

- If the activation is:
  - Positive, output +1
  - Negative, output -1
Multiclass Decision Rule

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

$$\text{activation}_w(x, y) = w_y \cdot f(x)$$

- Highest activation wins

$$y = \arg \max_y (\text{activation}_w(x, y))$$
The Multi-class Perceptron Alg.

- Start with zero weights
- Iterate training examples
  - Classify with current weights
    \[ y = \arg \max_y w_y \cdot f(x) = \arg \max_y \sum_i w_{y,i} \cdot f_i(x) \]
  - If correct, no change!
  - If wrong: lower score of wrong answer, raise score of right answer
    \[ w_y = w_y - f(x) \]
    \[ w_{y*} = w_{y*} + f(x) \]
Examples: Perceptron

- Separable Case

Examples: Perceptron

- Inseparable Case
Mistake-Driven Classification

- For Naïve Bayes:
  - Parameters from data statistics
  - Parameters: probabilistic interpretation
  - Training: one pass through the data

- For the perceptron:
  - Parameters from reactions to mistakes
  - Parameters: discriminative interpretation
  - Training: 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.

\[ \text{mistakes} < \frac{k}{\delta^2} \]
Problems with the Perceptron

- **Noise**: if the data isn’t separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)

- **Mediocre generalization**: finds a “barely” separating solution

- **Overtraining**: test / held-out accuracy usually rises, then falls
  - Overtraining is a kind of overfitting
Fixing the Perceptron

- Idea: adjust the weight update to mitigate these effects

- MIRA*: choose an update size that fixes the current mistake…

- … but, minimizes the change to $w$

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

$$w_{y^*} \cdot f(x) \geq w_y \cdot f(x) + 1$$

- The +1 helps to generalize

* Margin Infused Relaxed Algorithm
Minimum Correcting Update

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

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

\[ \min_{\tau} \|\tau f\|^2 \]

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

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

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

\[ w_y = w'_y - \tau f(x) \]

\[ w_y^* = w'_y^* + \tau f(x) \]

\[ \tau = 0 \]

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

\[ \min \text{ not } \tau = 0, \text{ or would not have made an error, so } \min \text{ will be where equality holds} \]
Maximum Step Size

- In practice, it’s also bad to make updates that are too large
  - Example may be labeled incorrectly
  - You may not have enough features
  - Solution: cap the maximum possible value of $\tau$ with some constant $C$

$$\tau^* = \min \left( \frac{(w_y' - w_{y*}') \cdot f + 1}{2f \cdot f}, C \right)$$

- Corresponds to an optimization that assumes non-separable data
- Usually converges faster than perceptron
- Usually better, especially on noisy data
Linear Separators

- Which of these linear separators is optimal?
Support Vector Machines

- Maximizing the margin: good according to intuition, theory, practice
- 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

MIRA

\[
\min_w \frac{1}{2} ||w - w'||^2 \\
w_y^* \cdot f(x_i) \geq w_y \cdot f(x_i) + 1
\]

SVM

\[
\min_w \frac{1}{2} ||w||^2 \\
\forall i, y \ w_y^* \cdot f(x_i) \geq w_y \cdot f(x_i) + 1
\]
Classification: Comparison

- **Naïve Bayes**
  - Builds a model from training data
  - Gives prediction probabilities
  - Strong assumptions about feature independence
  - One pass through data (counting)

- **Perceptrons / MIRA:**
  - Makes less assumptions about data
  - Mistake-driven learning
  - Multiple passes through data (prediction)
  - Often more accurate
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

\[ x = \text{“Apple Computers”} \]
Feature-Based Ranking

$x = \text{“Apple Computers”}$

\[ f(x, \text{Apple}) = [0.3 \ 5 \ 0 \ 0 \ldots] \]

\[ f(x, \text{Apple Inc.}) = [0.8 \ 4 \ 2 \ 1 \ldots] \]
Perceptron for Ranking

- Inputs $x$
- Candidates $y$
- Many feature vectors: $f(x, y)$
- One weight vector: $w$
  - Prediction:
    \[ y = \arg \max_y w \cdot f(x, y) \]
  - 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?
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?
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*

http://www.cs.cmu.edu/~zhuxj/courseproject/knndemo/KNN.html
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:
  \[ 1 = \langle 0.0 \ 0.0 \ 0.3 \ 0.8 \ 0.7 \ 0.1 \ldots 0.0 \rangle \]

- What’s the similarity function?
  - Dot product of two images vectors?
    \[ \text{sim}(x, x') = x \cdot x' = \sum_i x_i x_i' \]
  - 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 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**

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!
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!
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)

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

$$w_y = w_y - f(x)$$
$$w_{y^*} = w_{y^*} + f(x)$$
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:

\[ \Phi: \mathbf{x} \rightarrow \phi(\mathbf{x}) \]
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
Recap: Classification

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

- Clustering systems:
  - Unsupervised learning
  - Detect patterns in unlabeled data
    - E.g. group emails or search results
    - E.g. find categories of customers
    - E.g. detect anomalous program executions
  - Useful when don’t know what you’re looking for
  - Requires data, but no labels
  - Often get gibberish
Clustering

- Basic idea: group together similar instances
- Example: 2D point patterns

What could “similar” mean?
- One option: small (squared) Euclidean distance

\[
\text{dist}(x, y) = (x - y)^T(x - y) = \sum_i (x_i - y_i)^2
\]
K-Means

- An iterative clustering algorithm
  - Pick K random points as cluster centers (means)
  - Alternate:
    - Assign data instances to closest mean
    - Assign each mean to the average of its assigned points
  - Stop when no points’ assignments change
K-Means Example
K-Means as Optimization

- Consider the total distance to the means:
\[
\phi(\{x_i\}, \{a_i\}, \{c_k\}) = \sum_i \text{dist}(x_i, c_{a_i})
\]

- Each iteration reduces \( \phi \)

- Two stages each iteration:
  - Update assignments: fix means \( c \), change assignments \( a \)
  - Update means: fix assignments \( a \), change means \( c \)
Initialization

- **K-means is non-deterministic**
  - Requires initial means
  - It does matter what you pick!

- What can go wrong?

- Various schemes for preventing this kind of thing: variance-based split / merge, initialization heuristics
K-Means Getting Stuck

- A local optimum:

Why doesn’t this work out like the earlier example, with the purple taking over half the blue?
K-Means Questions

- Will K-means converge?
  - To a global optimum?

- Will it always find the true patterns in the data?
  - If the patterns are very very clear?

- Will it find something interesting?

- Do people ever use it?

- How many clusters to pick?
Agglomerative Clustering

- **Agglomerative clustering:**
  - First merge very similar instances
  - Incrementally build larger clusters out of smaller clusters

- **Algorithm:**
  - Maintain a set of clusters
  - Initially, each instance in its own cluster
  - Repeat:
    - Pick the two closest clusters
    - Merge them into a new cluster
    - Stop when there's only one cluster left

- Produces not one clustering, but a family of clusterings represented by a dendrogram
How should we define “closest” for clusters with multiple elements?

Many options
- Closest pair (single-link clustering)
- Farthest pair (complete-link clustering)
- Average of all pairs
- Ward’s method (min variance, like k-means)

Different choices create different clustering behaviors
Clustering Application

Top-level categories: supervised classification

Story groupings: unsupervised clustering