

Beyond binary classification

Subhransu Maji

CMPSCI 689: Machine Learning

19 February 2015

Administrivia

- ◆ Mini-project 1 posted
 - ▶ One of three
 - ▶ Decision trees and perceptrons
 - ▶ Theory and programming
 - ▶ Due Wednesday, March 04, 11:55pm **4:00pm**
 - Turn in a hard copy in the CS office
 - ▶ Must be done individually, but feel free to discuss with others
 - ▶ Start early ...

Today's lecture

- ◆ Learning with imbalanced data
- ◆ Beyond binary classification
 - ▶ Multi-class classification
 - ▶ Ranking
 - ▶ Collective classification

CMPSCI 689

Subhransu Maji (UMASS)

2/27

Learning with imbalanced data

- ◆ One class might be rare (E.g., face detection)
- ◆ Mistakes on the **rare** class cost more:
 - ▶ cost of misclassifying $y=+1$ is α (>1)
 - ▶ cost of misclassifying $y=-1$ is 1
- ◆ **Why?** we want is a better **f-score** (or **average precision**)

binary classification

$$\mathbb{E}_{(\mathbf{x}, y) \sim D} [f(\mathbf{x}) \neq y]$$

α -weighted binary classification

$$\mathbb{E}_{(\mathbf{x}, y) \sim D} [\alpha^{y=1} f(\mathbf{x}) \neq y]$$

Suppose we have an algorithm to train a binary classifier,
can we use it to train the alpha weighted version?

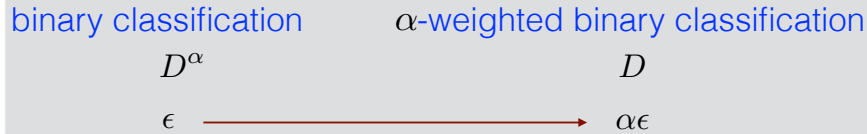
Training by sub-sampling

- ◆ **Input:** D, α **Output:** D^α
- ◆ **While true**
 - Sample $(\mathbf{x}, y) \sim D$
 - Sample $t \sim \text{uniform}(0, 1)$
 - If $y > 0$ or $t < 1/\alpha$
 - return (\mathbf{x}, y)

We have sub-sampled the negatives by $1/\alpha$

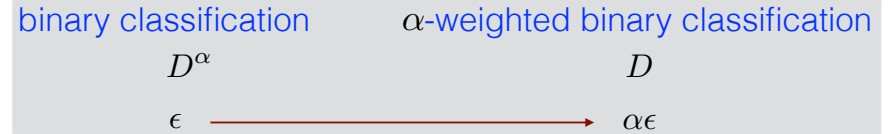
sub-sampling algorithm

Claim



Proof of the claim

$$\begin{aligned}
 \text{Error on } \mathcal{D} &= \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{D}}[\ell^\alpha(\hat{y}, y)] \\
 &= \sum_{\mathbf{x}} (D(\mathbf{x}, +1)\alpha[\hat{y} \neq 1] + D(\mathbf{x}, -1)[\hat{y} \neq -1]) \\
 &= \alpha \left(\sum_{\mathbf{x}} \left(D(\mathbf{x}, +1)[\hat{y} \neq 1] + \frac{1}{\alpha} D(\mathbf{x}, -1)[\hat{y} \neq -1] \right) \right) \\
 &= \alpha \left(\sum_{\mathbf{x}} (D^\alpha(\mathbf{x}, +1)[\hat{y} \neq 1] + D^\alpha(\mathbf{x}, -1)[\hat{y} \neq -1]) \right) \\
 &= \alpha\epsilon
 \end{aligned}$$



Modifying training

- ◆ To train simply —
 - Subsample negatives and train a binary classifier.
 - Alternatively, supersample positives and train a binary classifier.
 - Which one is better?
- ◆ For some learners we don't need to keep copies of the positives
 - **Decision tree**
 - Modify accuracy to the weighted version
 - **kNN classifier**
 - Take weighted votes during prediction
 - **Perceptron?**

Overview

- ◆ Learning with imbalanced data
- ◆ Beyond binary classification
 - **Multi-class classification**
 - **Ranking**
 - **Collective classification**

Multi-class classification

- ◆ Labels are one of K different ones.
- ◆ Some classifiers are inherently multi-class —
 - ▶ **kNN classifiers**: vote among the K labels, pick the one with the highest vote (break ties arbitrarily)
 - ▶ **Decision trees**: use multi-class histograms to determine the best feature to splits. At the leaves predict the most frequent label.
- ◆ **Question**: can we take a binary classifier and turn it into multi-class?

One-vs-all (OVA) classifier

- ◆ Train K classifiers, each to distinguish **one class from the rest**
- ◆ Prediction: pick the class with the **highest score**:

$$i \leftarrow \arg \max f_i(\mathbf{x}) \quad \text{score function}$$

◆ Example

- ▶ **Perceptron**: $i \leftarrow \arg \max \mathbf{w}_i^T \mathbf{x}$
 - May have to calibrate the weights (e.g., **fix the norm to 1**) since we are comparing the scores of classifiers
 - In practice, doing this right is tricky when there are a large number of classes

One-vs-one (OVO) classifier

- ◆ Train $K(K-1)/2$ classifiers, each to distinguish **one class from another**
- ◆ Each classifier votes for the winning class in a pair
- ◆ The class with most votes wins

$$i \leftarrow \arg \max \left(\sum_j f_{ij}(\mathbf{x}) \right)$$

$$f_{ji} = -f_{ij}$$

◆ Example

- ▶ **Perceptron**: $i \leftarrow \arg \max \left(\sum_j \text{sign}(\mathbf{w}_{ij}^T \mathbf{x}) \right)$ $\mathbf{w}_{ji} = -\mathbf{w}_{ij}$
 - Calibration is **not an issue** since we are taking the sign of the score

Directed acyclic graph (DAG) classifier

- ◆ DAG SVM [Platt et al., NIPS 2000]
 - ▶ Faster testing: $O(K)$ instead of $O(K(K-1)/2)$
 - ▶ Has some theoretical guarantees

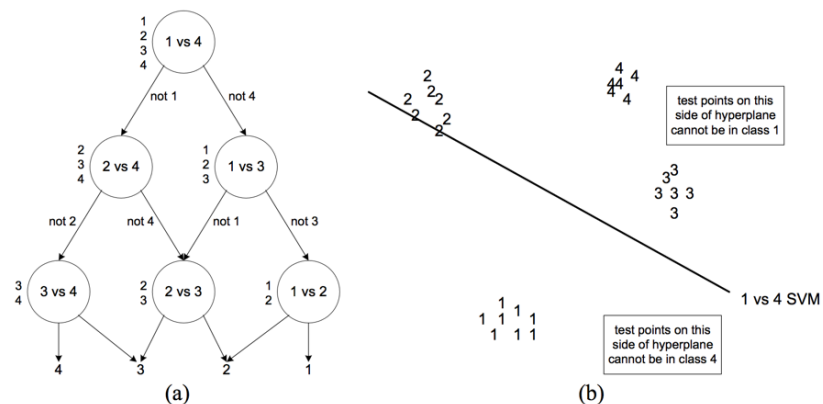
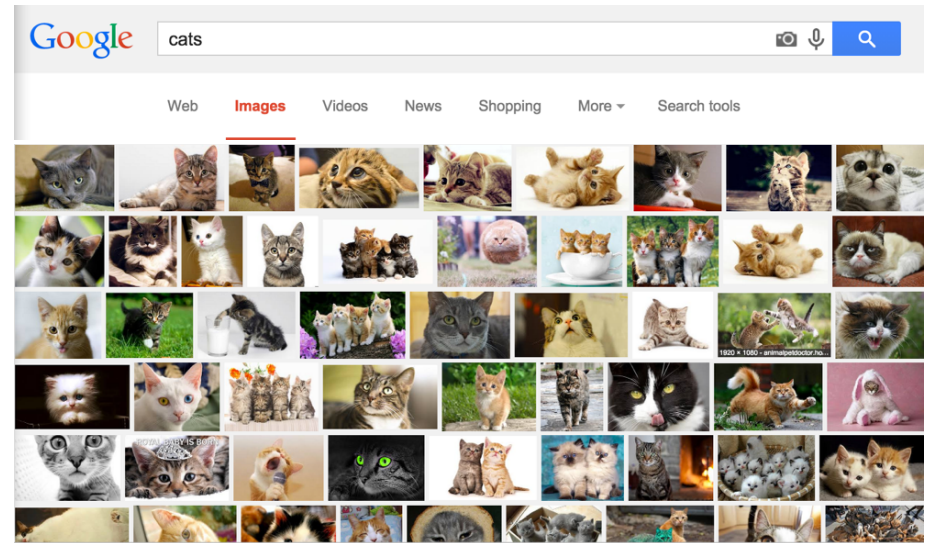


Figure from Platt et al.

Overview

- ◆ Learning with imbalanced data
- ◆ Beyond binary classification
 - ▶ Multi-class classification
 - ▶ Ranking
 - ▶ Collective classification

Ranking



Ranking

- ◆ Input: [query](#) (e.g. “cats”)
- ◆ Output: [a sorted list of items](#)

- ◆ How should we measure [performance](#)?
- ◆ The [loss function](#) is trickier than in the binary classification case
 - ▶ Example 1: [All items in the first page should be relevant](#)
 - ▶ Example 2: [All relevant items should be ahead of irrelevant items](#)

Learning to rank

- ◆ For simplicity lets assume we are learning to rank for a given query.
- ◆ [Learning to rank](#):
 - ▶ Input: [a list of items](#)
 - ▶ Output: [a function that takes a set of items and returns a sorted list](#)

- ◆ [Approaches](#)
 - ▶ [Pointwise approach](#):
 - Assumes that each document has a numerical score.
 - Learn a [model](#) to predict the score (e.g. [linear regression](#)).
 - ▶ [Pairwise approach](#):
 - Ranking is approximated by a [classification problem](#).
 - Learn a [binary classifier](#) that can tell which item is better given a pair.

Naive rank train

◆ Create a dataset with binary labels

- Initialize: $D \leftarrow \phi$
- For every i and j such that, $i \neq j$
 - If item i is *more* relevant than j
 - Add a **positive** point: $D \leftarrow D \cup (\mathbf{x}_{ij}, +1)$
 - If item i is *less* relevant than j
 - Add a **negative** point: $D \leftarrow D \cup (\mathbf{x}_{ij}, -1)$

$\mathbf{x}_{ij} \leftarrow$ features for comparing item i and j

◆ Learn a binary classifier on D

◆ Ranking

- Initialize: $score \leftarrow [0, 0, \dots, 0]$
 - For every i and j such that, $i \neq j$
 - Calculate **prediction**: $y \leftarrow f(\hat{\mathbf{x}}_{ij})$
 - Update **scores**: $score_i = score_i + y$ $score_j = score_j - y$
- ranking $\leftarrow \text{arg sort}(score)$

Problems with naive ranking

◆ Naive rank train works well for bipartite ranking problems

- Where the goal is to predict whether an item is relevant or not. There is no notion of an item being *more* relevant than another.

◆ A better strategy is to account for the positions of the items in the list

◆ Denote a ranking by: σ

- If item u appears before item v , we have: $\sigma_u < \sigma_v$

◆ Let the space of all permutations of M objects be: Σ_M

◆ A ranking function maps M items to a permutation: $f: \mathcal{X} \rightarrow \Sigma_M$

◆ A cost function (omega)

- The cost of placing an item at position i at j : $\omega(i, j)$

◆ Ranking loss: $\ell(\sigma, \hat{\sigma}) = \sum_{u \neq v} [\sigma_u < \sigma_v][\hat{\sigma}_v < \hat{\sigma}_u] \omega(u, v)$

$$\omega\text{-ranking: } \min_f \mathbb{E}_{(\mathcal{X}, \sigma) \sim \mathcal{D}} [\ell(\sigma, \hat{\sigma})], \text{ where } \hat{\sigma} = f(\mathcal{X})$$

ω -rank loss functions

◆ To be a valid loss function ω must be:

- **Symmetric**: $\omega(i, j) = \omega(j, i)$
- **Monotonic**: $\omega(i, j) \leq \omega(i, k)$ if $i < j < k$ or $k < j < i$
- **Satisfy triangle inequality**: $\omega(i, j) + \omega(j, k) \geq \omega(i, k)$

◆ Examples:

- **Kemeny loss**:

$$\omega(i, j) = 1, \text{ for } i \neq j$$

- **Top-K loss**:

$$\omega(i, j) = \begin{cases} 1 & \text{if } \min(i, j) \leq K, i \neq j \\ 0 & \text{otherwise} \end{cases}$$

ω -rank train

◆ Create a dataset with binary labels

- Initialize: $D \leftarrow \phi$
- For every i and j such that, $i \neq j$
 - If $\sigma_i < \sigma_j$ (item i is more relevant)
 - Add a **positive** point: $D \leftarrow D \cup (\mathbf{x}_{ij}, +1, \omega(i, j))$
 - If $\sigma_i > \sigma_j$ (item j is more relevant)
 - Add a **negative** point: $D \leftarrow D \cup (\mathbf{x}_{ij}, -1, \omega(i, j))$

$\mathbf{x}_{ij} \leftarrow$ features for comparing item i and j

◆ Learn a binary classifier on D (each instance has a weight)

◆ Ranking

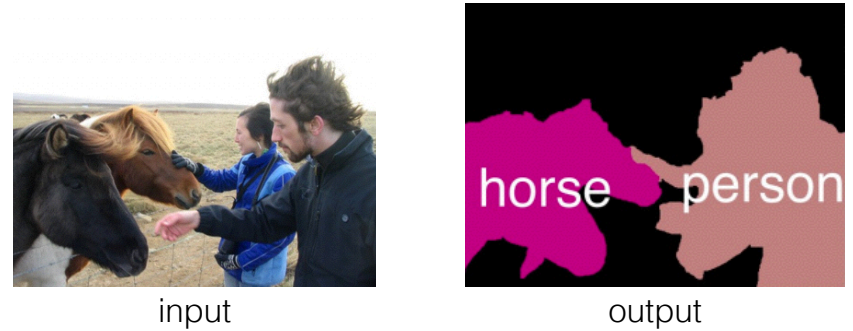
- Initialize: $score \leftarrow [0, 0, \dots, 0]$
 - For every i and j such that, $i \neq j$
 - Calculate **prediction**: $y \leftarrow f(\hat{\mathbf{x}}_{ij})$
 - Update **scores**: $score_i = score_i + y$ $score_j = score_j - y$
- ranking $\leftarrow \text{arg sort}(score)$

Overview

- ◆ Learning with imbalanced data
- ◆ Beyond binary classification
 - Multi-class classification
 - Ranking
 - Collective classification

Collective classification

- ◆ Predicting **multiple correlated variables**



$(\mathbf{x}, k) \in \mathcal{X} \times [K]$ $\mathcal{G}(\mathcal{X}, k)$ be the set of all graphs
 features labels

objective $f : \mathcal{G}(\mathcal{X}) \rightarrow \mathcal{G}([K])$ $\mathbb{E}_{(V,E) \sim \mathcal{D}} [\sum_{v \in V} (\hat{y}_v \neq y_v)]$

Collective classification

- ◆ Predicting **multiple correlated variables**



$\hat{y}_v \leftarrow f(\mathbf{x}_v)$

independent predictions can be noisy

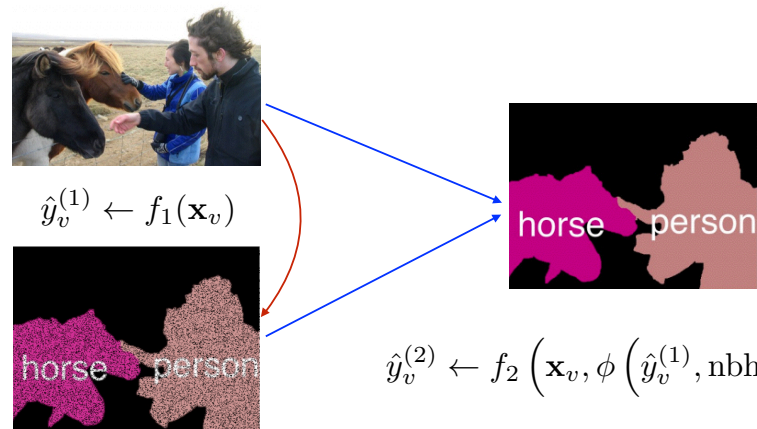
labels of nearby vertices as features

$\mathbf{x}_v \leftarrow [\mathbf{x}_v, \phi([K], \text{nbhd}(v))]$

E.g., histogram of labels in a 5x5 neighborhood

Stacking classifiers

- ◆ Train a two classifiers
- ◆ First one is trained to predict output from the **input**
- ◆ Second is trained on the **input** and the **output of first classifier**

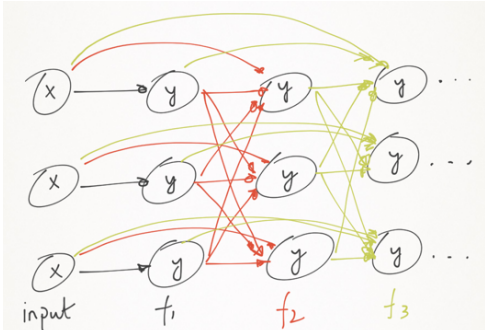


$\hat{y}_v^{(1)} \leftarrow f_1(\mathbf{x}_v)$

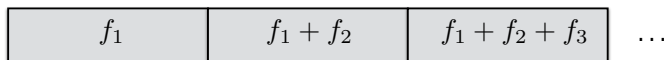
$\hat{y}_v^{(2)} \leftarrow f_2(\mathbf{x}_v, \phi(\hat{y}_v^{(1)}, \text{nbhd}(v)))$

Stacking classifiers

- ◆ Train a stack of N classifiers
- ◆ i^{th} classifier is trained on the **input + output of the previous $i-1$ classifiers**



- ◆ **Overfitting is an issue:** the classifiers are accurate on training data but on not on test data leading to a cascade of overconfident classifiers
- ◆ Solution: **held-out data**



Summary

- ◆ Learning with imbalanced data
 - Implicit and explicit sampling can be used to train binary classifiers for the weighted loss case
- ◆ Beyond binary classification
 - **Multi-class classification**
 - Some classifiers are inherently multi-class
 - Others can be combined using: one-vs-one, one-vs-all methods
 - **Ranking**
 - Ranking loss functions to capture distance between permutations
 - Pointwise and pairwise methods
 - **Collective classification**
 - Stacking classifiers trained with held-out data

Slides credit

- ◆ Some slides are adapted from CIML book by Hal Daume
- ◆ Images for collective classification are from the PASCAL VOC dataset
 - <http://pascallin.ecs.soton.ac.uk/challenges/VOC/>
- ◆ Some of the discussion is based on Wikipedia
 - http://en.wikipedia.org/wiki/Learning_to_rank