CSci 4270 and 6270
Computational Vision, Spring 2021
Lecture 13: Introduction to Machine Learning for Computer Vision
March 11, 2021

(Review March 15)

Today

- Complete Lecture 12 notes: two-image mosaics and multi-image mosaics.
- Discuss HW 4
- Introduction to machine learning for computer vision

Next Few Weeks — Slight Rearrangement

- Monday, 3/15: SVMs and Detection
- Thursday, 3/18: Neural networks
- Monday, 3/22: Convolutional neural networks
- Thursday, 3/25: Recognition, part 1
- Monday, 3/29: Recognition, part 2
- Thursday, 4/01: Data sets
- Monday, 4/05: Bias and social issues

Aside: Importance of Modeling

- Ideal steps:
  1. Write down your model describing as precisely as possible your understanding of the problem you need to solve and the input data you must use to solve the problem.
  2. Develop an algorithm to solve the model (estimate its parameters for a given input data set) as efficiently and effectively as possible.
  3. Implement and test
  4. Refine model; refine algorithm; refine implementation; repeat.
- Model should not depend on the algorithm
- Algorithm should solve the problem without changing the model.
- We'll brainstorm the model for the bat counting problem in class.
- We'll think about modeling asssumptions as we proceed.

assumptions

small bats

simple background → uniform

only non background were bats

no fixed shape to bats

bats contiguous

bats separated

models

background constant intensity

$\bar{r}, \bar{g}, \bar{b} \Rightarrow$ gray

estimate σ

bats were stats sign dark

algs allowed simple

- avg
- std
→ thresh
→ morph
→ merging of nearby bats

More sophisticated:

slowly varying intensity

bats locally darker

modeled additional objects

Learning in Vision

Note that much of this is adapted from Chapter 5 of Szeliski's book.

Four generic "types" of learned computer vision solutions:

1. Non-learning:

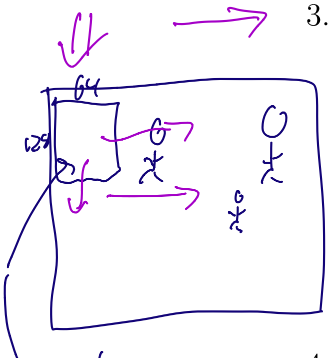
   images → hand-crafted features → hand-crafted algorithms → output

   - Example: Estimating **F** and **H** from SIFT keypoints and descriptor matching

2. Shallow learning (version A):

   images → learned features → hand-crafted CV algorithms → output

   - Example: Estimating F and H from learned alternatives to SIFT keypoints and descriptors.



3. Shallow learning (version B):

   images → hand-crafted features → learning algorithm → output

   - Example B1: sliding window detectors using Histogram of Oriented Gradients and linear support vector machines. (See Lecture 14.)
   - Example B2: texture descriptors and color histograms combined with a random forest classifier to classify disease states of images of tissue samples.

Compute "SIFT on steroids" ⇒ 5K descriptor

4. Deep (end-to-end) learning:

   images → learning algorithm → output

   - Example: feed forward neural networks for image classification
     - Car, tree, forest, grass, lion, tiger, etc...

Start with a Model

- Model

$$f(\mathbf{x}; \mathbf{w})$$

data ≈ parameters to be learned

where

  - $\mathbf{x}$ is the input data — which could be an image, or a set of feature (descriptor) vectors,
  - $\mathbf{w}$ is the weight/parameter vector whose values are to be learned,
  - f is the function (could be vector valued) mapping the input to the output, guided by the weights. This is the "algorithm"

- Examples of f:

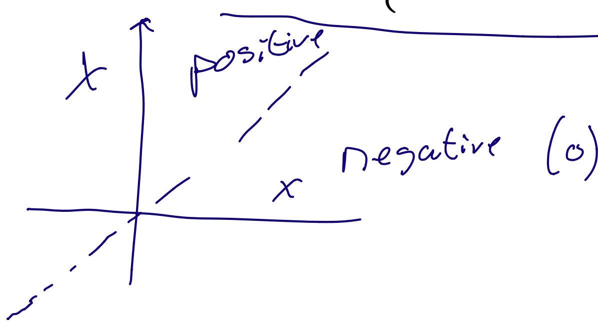
params linear regression

  - Linear regression
  - Estimation of **H**, **F** from keypoint matches

w is H or F params plus labeling of inliers/outliers

  - Binary classifier:

$$f(\mathbf{x}, \mathbf{w}) = \begin{cases} 1 & \text{if } \mathbf{x}^\top \mathbf{w} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$



Types of ML

Measurement "true label"

- Supervised: given a set of N examples $\{(\mathbf{x}_i, y_i)\}, i \in 1, \ldots, N,$ where $f(\mathbf{x}_i; \mathbf{w})$ should be y_i, find the best estimate of the parameters $\mathbf{w}$.

  For example,

  x_i are images y_i are cat/dog label

  - Given many (thousands) of images showing cats and many thousands of images showing dogs.
  - f is a classifier 1 is dog 0 (or -1) if cat
  - Learn the set of $\mathbf{w}$ that best allows f to determine if a new image shows a cat or a dog.

- Unsupervised: given a set of examples $\{\mathbf{x}\}$, find the best estimate of the parameters $\mathbf{w}$. not any y_i's

  - Example: given binary images of hand-written digits (MNIST), with no labels, can you assign each to a cluster and have the clusters be semantically meaningful? Ideally, each cluster would only contain binary images from all the same digits.

28x28

→ 764 comp binary vect

- Unsupervised learning is often used as a pre-processing or post-processing step with supervised learning.

cluster

10

Supervised Learning: Training and Loss; Test

- Given training data set $\{(\mathbf{x}_i, y_i)\}, i \in 1, \ldots, N$, the error function to be minimized is the "risk":

$$E(\mathbf{w}) = \sum_{i=1}^{N} L(y_i, f(\mathbf{x}; \mathbf{w})).$$

measure the cost of an error

- Here L is the "loss" function, which could be as simple as the square error for regression problems such as line fitting:

$$L(y_i, f(\mathbf{x}; \mathbf{w})) = (y_i - f(\mathbf{x}; \mathbf{w}))^2$$

or binary cross-entropy for binary labeling problems

$$L(y_i, f(\mathbf{x}; \mathbf{w})) = -[y_i \cdot \log(f(\mathbf{x}; \mathbf{w})) + (1 - y_i) \log(1 - f(\mathbf{x}; \mathbf{w}))]$$

$y_i = 0$ or 1

when true label is 1

L is

$-\log(f(x; w))$

$y_i = 0$

L is

$-\log(1 - f(x_i; w))$

- The parameter vector that minimizes E is the set of "learned parameters",

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}}\, E(\mathbf{w})$$

This is the "training" phase.

- The "test" phase occurs when the learned model is actually used. In particular, new input vector $\mathbf{x}$,

$$f(\mathbf{x}; \mathbf{w}^*)$$

  is the model "prediction".

- Note that this may be a continuous (regression) or binary (classification) value.

- Most of our discussion will be about classifiers.

- An important concern with many classifiers is how well they can be extended to multiple (more than two) classes.

Example: Least-Squares Regression

- In lecture we'll look at
  - The model
  - The parameters
  - The data
  - Training
  - Test and prediction

$f(\vec{x}; \vec{w}) = w^T x$

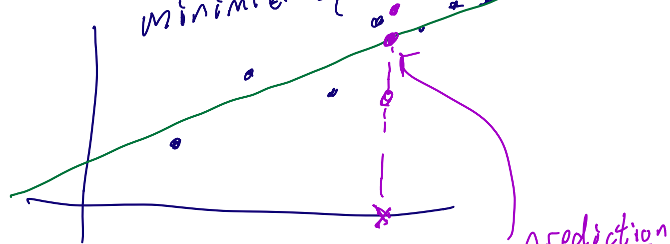
could be offset "bias" b

w

$\vec{x}_i , \vec{y}_i$

- We'll also think about how to make this into a classifier e.g. (just making this up) is a tree taller than averge based on its age and circumference?

estimate w that minimize sq error



new x

prediction $f(x; w^*) = w^{*T} x$

Classification

if given y as well as x

is y "above" $f(x, w^*)$

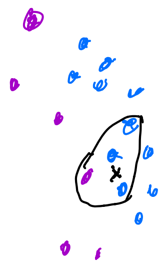
The Simplest Classifier: K Nearest Neighbors

are +1
are −1

- The training samples and their labels become the "learned" vector:
  - The value of k is an example of what's often called a "meta parameter"
- $f(\mathbf{x};\mathbf{w})$

  test

  1. Finds the k nearest $\mathbf{x}_i$ to $\mathbf{x}$, and then
  2. Finds the most common label among the associated y_i's as its classification label (decision).



- Choice of k is made empirically:
  - Too small and jagged decision boundaries result.
  - Too large and details of decision boundaries are lost.

3 blues
1 purple

K=4

so
Label is
blue

- High dimensionality is a crucial challenge:
  - Sparse
  - Efficient search is difficult
- K-NN is used frequently in practice, although not directly on image data. Instead, it is often used in the (relatively) low dimensional "embedding space" (also called a "latent space") produced by a deep neural network.
- Easily extends to multiclass problems.

Support Vector Machines

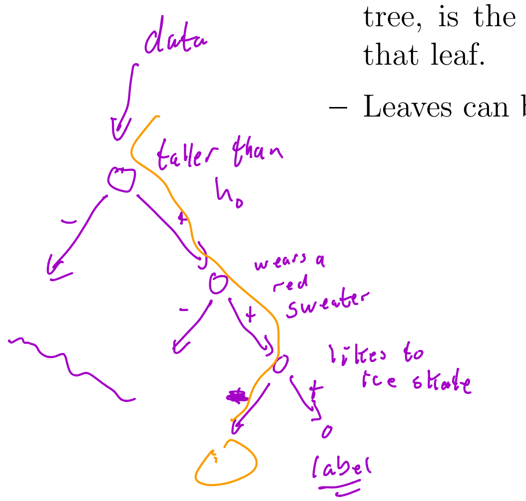
- Linear decision model:

$$f(\mathbf{x}; \mathbf{w}) = \begin{cases} 1 & \text{if } \mathbf{x}^\top \mathbf{w} \geq 0 \\ -1 & \text{if } \mathbf{x}^\top \mathbf{w} < 0 \end{cases}$$

- $\{(\mathbf{x}_i, y_i)\}, i \in 1, \ldots, N$ with $y_i \in \{-1, 1\}$.
- $\mathbf{w}$ chosen to maximize the separation (margin) between positive and negative samples, balancing this against misclassification error.
- Easily generalized to non-linear models using "kernels".
- Harder to extend to multiclass problems.
- We will talk about linear SVM's in Lecture 14.

Random Forests

- Forest of random decision trees.
- Decision trees have a classifier at each node:
  - Test values, $\mathbf{x}$ are sent left or right at a node depending on whether the node's classifier decision is positive or negative.
  - The label of a leaf node, which is the final overall decision of the tree, is the majority label of the training data values that reach that leaf.
  - Leaves can be interpreted probabilisitically as well.



training

after building send all data through tree and gather at each leaf

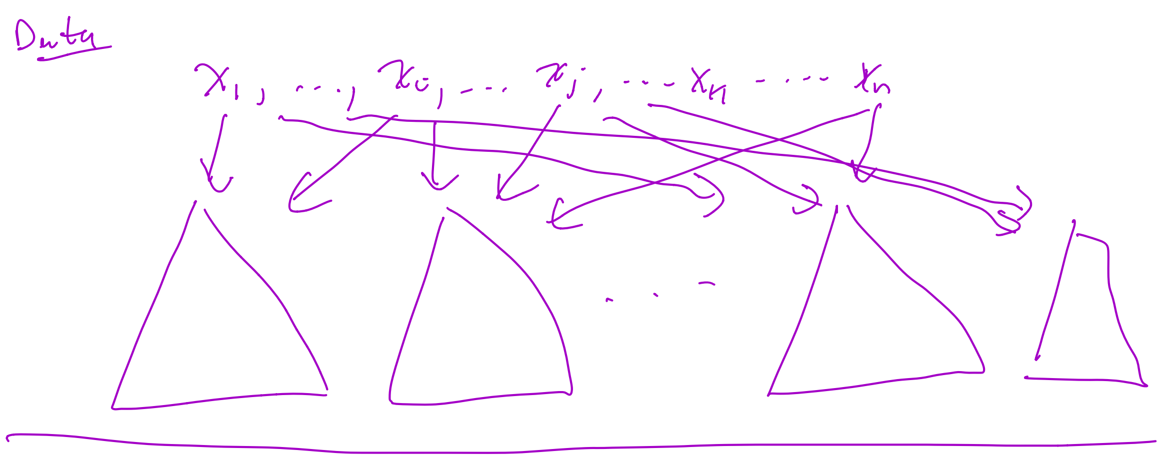
label at leaf is majority label of the data that arrived there

$C_1 = 2$
$C_2 = 7$
$C_3 = 1$

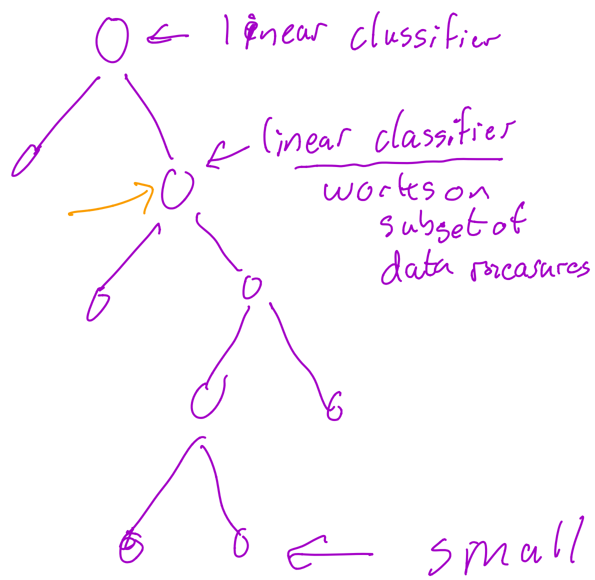
label is class 2
prob 70%

- Training a random forest, one random decision tree at a time:

  1. Each decision tree is trained on a randomly selected sample set of the training data. The training data stays the same for the whole tree.
     - Each training data instance $\mathbf{x}_i, y_i$ participates in the training of a significant number of these trees.
  2. Each node of each decision tree is a trained linear classifier, with positive classified training samples sent to the left subtree and negatively classified training samples sent to the right.
  3. Typically, each node's classifier is trained on a randomly chosen small subset of the feature vectors. For example, it might make a decision on just color measurements that are in the vector.
  4. Leaf nodes are formed when a small enough number of samples are in a child node formed after a split.



Training single tree



select classifier to split as homogeneously as possible.

ex: goal decide if scene contains man-made objects

data: measures of color
texture
position in image
....

← small enough sample size

Finally run all data through tree and gather stats at leaves.

- Test / prediction for data vector $\mathbf{x}$:
  - $\mathbf{x}$ is sent through each decision tree to a leaf node.
  - Decisions across all leaf nodes are aggregated into final decision.
- Easily extends to multiple classes

one leaf node
per data item
per tree

Final Note on Supervised Learning: Discriminative vs. Generative

- Discriminative classifiers learn decision boundaries and the probability

$$p(y \mid \mathbf{x})$$

  They attempt to find the most likely y given the data $\mathbf{x}$. There is usually no explicit probability function constructed.

- Generative models explicitly model the joint distribution

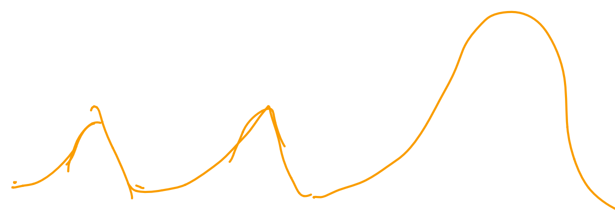
$$p(y, \mathbf{x}),$$

  they can generate samples from this distribution, and they reason using Bayes theorem.

- We will focus almost exclusively on discriminative models.

Unsupervised Learning

- Uses:
  - Summarize data in low dimensional space or on manifolds
  - Group data into (hopefully) semantically meaningful clusters.
  - Fit complex distributions to model the data
- Summarization:
  - PCA
  - Manifold learning
  - tSNE
- Clustering:
  - K-means
  - Agglomerative
- Data modeling:
  - Gaussian mixture models



Looking Ahead

- We will discuss SVMs as part of an introduction to detection during the next class.
- Most of our subsequent focus will be on neural networks as classifiers.
- Many of the other machine learning algorithms introduced (or just mentioned) here are sometimes competitors to neural networks and sometimes tools to be used with neural networks.
- We will discuss a variety of other topics along the way, including the all-important topic of data, data sources and bias.