CSE 250B: Machine Learning

Lecture 0: Overview

Machine learning versus Algorithms

In both fields, the goal is to develop

procedures that exhibit a desired input-output behavior.

- **Algorithms**: the input-output mapping can be precisely defined.
  Input: Graph G.
  Output: MST of G.
- **Machine learning**: the mapping cannot easily be made precise.
  Input: Picture of an animal.
  Output: Name of the animal.

Instead, we simply provide examples of (input,output) pairs and ask the machine to *learn* a suitable mapping itself.

Inputs and outputs

Basic terminology:

- The input space, $\mathcal{X}$.
  E.g. 32×32 RGB images of animals.
- The output space, $\mathcal{Y}$.
  E.g. Names of 100 animals.

x:

y: "bear"

After seeing a bunch of examples (x, y), pick a mapping

$$f : \mathcal{X} \to \mathcal{Y}$$

that accurately replicates the input-output pattern of the examples.

Learning problems are often categorized according to the type of *output space*: (1) discrete, (2) continuous, (3) probability values, or (4) more general structures.

Discrete output space: classification

Binary classification:

- Spam detection
  $\mathcal{X} = \{\text{email messages}\}$
  $\mathcal{Y} = \{\text{spam}, \text{not spam}\}$
- Credit card fraud detection
  $\mathcal{X} = \{\text{descriptions of credit card transactions}\}$
  $\mathcal{Y} = \{\text{fraudulent}, \text{legitimate}\}$

Multiclass classification:

- Animal recognition
  $\mathcal{X} = \{\text{animal pictures}\}$
  $\mathcal{Y} = \{\text{dog}, \text{cat}, \text{giraffe}, \ldots\}$
- News article classification
  $\mathcal{X} = \{\text{news articles}\}$
  $\mathcal{Y} = \{\text{politics}, \text{business}, \text{sports}, \ldots\}$

Continuous output space: regression

- A parent's concerns
  How cold will it be tomorrow morning?
  $\mathcal{Y} = [-273, \infty)$
- For the asthmatic
  Predict tomorrow's air quality (max over the whole day)
  $\mathcal{Y} = [0, \infty)$ (< 100: okay, > 200: dangerous)
- Insurance company calculations
  In how many years will this person die?
  $\mathcal{Y} = [0, 200]$

What are suitable predictor variables ($\mathcal{X}$) in each case?

Conditional probability functions

Here $\mathcal{Y} = [0, 1]$ represents probabilities.

- Dating service
  What is the probability these two people will go on a date if introduced to each other?
  If we modeled this as a classification problem, the binary answer would basically always be "no". The goal is to find matches that are slightly less unlikely than others.
- Credit card transactions
  What is the probability that this transaction is fraudulent?
  The probability is important, because – in combination with the amount of the transaction – it determines the overall risk and thus the right course of action.

Structured output spaces

The output space consists of structured objects, like sequences or trees.

Dating service

Input: description of a person
Output: rank-ordered list of all possible matches

$\mathcal{Y}$ = space of all permutations

Example:
x = *Tom*
y = (*Nancy*, *Mary*, *Chloe*, ...)

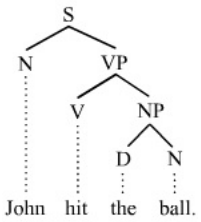
Language processing

Input: English sentence
Output: parse tree showing grammatical structure

$\mathcal{Y}$ = space of all trees

Example:
x = "John hit the ball"

$y =$

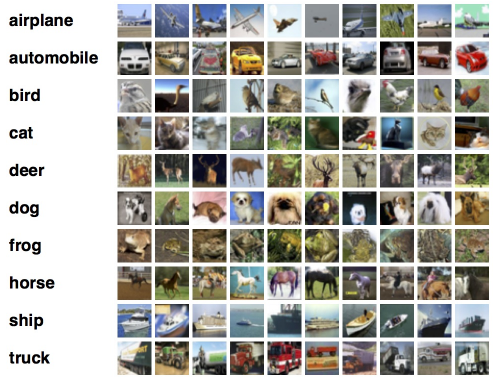


Course outline

1. Nonparametric methods
2. Classification using parametrized models
3. Combining classifiers
4. Representation learning

Nonparametric methods: nearest neighbor

Training set: a collection of (x, y) pairs:

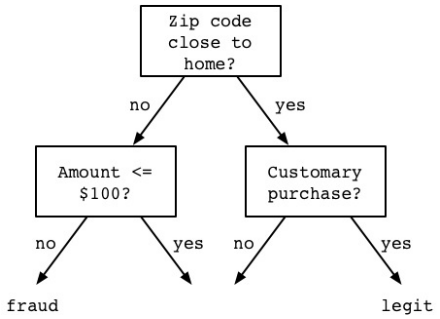


Given any x, find its nearest neighbor in the training set and predict that neighbor's y value.

Issues: (1) What distance function? (2) How to speed up search?

Nonparametric methods: decision tree

Credit card fraud detection: use training data to build a tree classifier



What do nearest neighbor and decision trees have in common?

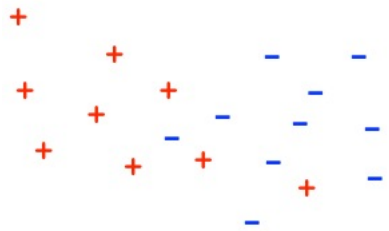
- Unbounded in size
- Can model arbitrarily complex functions

They are *nonparametric methods*.

Classification with parametrized models

Classifiers with a fixed number of parameters can represent a limited set of functions. Learning a model is about picking a good approximation.

Typically the x's are points in d-dimensional Euclidean space, $\mathbb{R}^d$:



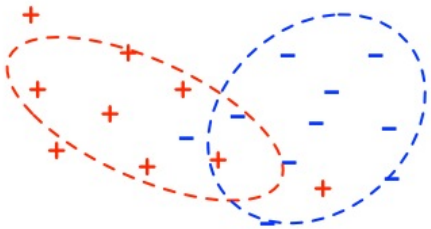
Two ways to classify:

- *Generative*: model the individual classes.
- *Discriminative*: model the decision boundary between the classes.

Generative models

Fit a probability distribution – like a multivariate Gaussian – to each class. Thereafter use this *summary* rather than the data points themselves.

To classify a new point: find the most probable class.



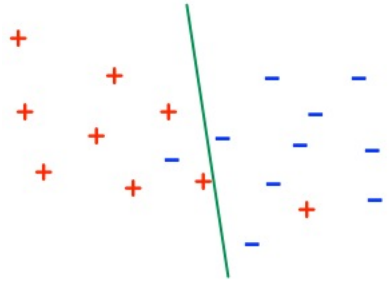
Examples: Naive Bayes, Fisher discriminant.

Under the hood: Bayes' rule, linear algebra (eigenvalues, eigenvectors).

Discriminative models

Approximate the boundaries between classes by simple – e.g. linear – functions.

To classify a new point: figure out which side of the boundary it lies on.



Examples: support vector machine, logistic regression.

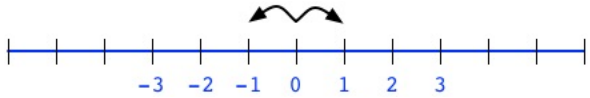
Under the hood: convex duality, optimization.

Generalization theory

- Complex, e.g. nonparametric, classifiers require a lot of training data to learn accurately.
- Simple, e.g. linear, classifiers require less.

What is the right notion of complexity? Are there formulas for how much data is enough? The answers are based on *large deviation theory*.

Example: *Symmetric random walk.*
A drunken man starts at the origin and at each time step either takes a step to the right or a step to the left. Where is he after n steps?

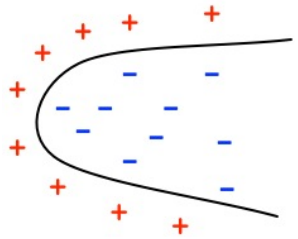


Answer: Not far from where he started! Most likely in $[-c\sqrt{n}, c\sqrt{n}]$ (for some constant c), and all positions in this interval are about equally likely.
Under the hood: probability theory.

Richer classifiers via the kernel trick

We are good at finding linear classifiers in Euclidean space. But what if:

- The boundary between classes is far from linear?
  Example: quadratic, or higher-order polynomial, or even stranger.



- The data aren't even vectors of numbers?
  Example: documents, DNA sequences, parse trees.

The *kernel trick* handles these scenarios seamlessly, by mapping the data to a suitable Euclidean space in which linear classification is possible!

Richer output spaces

Many classification methods were developed for the binary (two-label) case. Usually the output space is larger than this.

- $\mathcal{Y}$ = several classes.
  Examples:
  x = image, y = name of object in image
  x = news article, y = category (sports, politics, business, ...)
- $\mathcal{Y}$ = structured objects.
  Examples:
  x = sentence in Swahili, y = transcription into English
  x = sentence in English, y = parse tree

Extend binary classification to handle such cases!

Under the hood: error-correcting codes, dynamic programming.

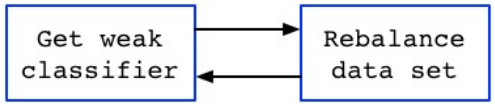
Composing simple classifiers

A common situation in classifier learning:

Easy to find **weak classifiers** – *not very accurate, but better than random*

To increase accuracy, *compose* weak classifiers.

Example: *boosting*.



Final classifier is a linear combination of all these weak classifiers.

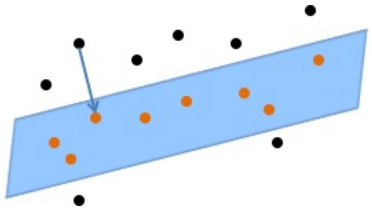
Generically improve the performance of *any* kind of classifier!

Representation learning

A handful of key primitives:

1. Dimensionality reduction and denoising.

   Given data in high-dimensional Euclidean space, project to a low-dimensional linear subspace while retaining as much of the signal as possible.



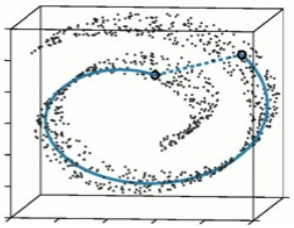
2. Embedding and manifold learning.
3. Metric learning.

Representation learning

A handful of key primitives:

1. Dimensionality reduction and denoising.
2. Embedding and manifold learning.

   Given data that lie in a non-Euclidean space, find an embedding into Euclidean space that preserves as much of the geometry as possible.



3. Metric learning.

Representation learning

A handful of key primitives:

1. Dimensionality reduction and denoising.
2. Embedding and manifold learning.
3. Metric learning.

   Given data with only vague positional information, impose an Euclidean geometry that is suitable for classification.

   Example: $\mathcal{X} = \{\text{a collection of } m \text{ books}\}$.

   A user supplies $\binom{m}{2}$ similarity ratings, such as:

   ("Pride and Prejudice", "Great Expectations"): similar

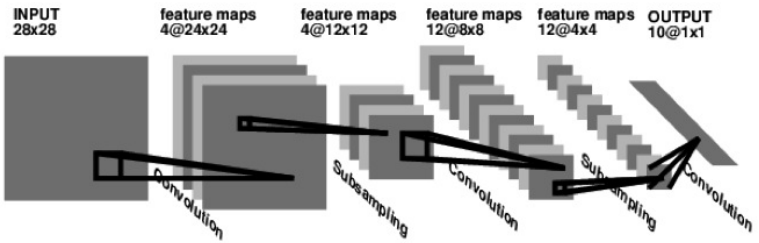
   ("Hamlet", "Great Expectations"): dissimilar

   Represent each book by a vector, respecting these ratings.

Under the hood: linear algebra, semidefinite programming.

Deep learning

Multi-layer neural nets achieve state-of-the-art performance across a range of benchmark problems in natural language processing, speech, and vision.



Under the hood: stochastic gradient descent, dictionary learning, autoencoders.

Course outline

1. Nonparametric methods
2. Classification using parametrized models
   - Generative models
   - Discriminative models
   - Richer decision boundaries using the kernel trick
   - Richer output spaces
   - Generalization theory
3. Combining classifiers
   - Boosting, bagging, and random forests
   - Online learning
4. Representation learning
   - Linear projection
   - Embeddings
   - Metric learning
   - Deep learning

Class details

Website: `http://www.cs.ucsd.edu/~dasgupta/250B`.

Grading:

- Regular homeworks: 50%.
- Two midterms: 25% each.

Nearest neighbor classification

CSE 250B

Nearest neighbor classification

Given a labeled training set $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$.

Example: the MNIST data set of handwritten digits.

To classify a new instance x:

- Find its nearest neighbor amongst the $x^{(i)}$
- Return $y^{(i)}$

The data space

We need to choose a distance function.

Each image is 28×28 grayscale.
One option: Treat images as 784-dimensional vectors, and use Euclidean (ℓ_2) distance:

$$\|x - x'\| = \sqrt{\sum_{i=1}^{784} (x_i - x_i')^2}.$$

Summary:

- Data space $\mathcal{X} = \mathbb{R}^{784}$ with ℓ_2 distance
- Label space $\mathcal{Y} = \{0, 1, \ldots, 9\}$

Performance on MNIST

Training set of 60,000 points.

- What is the error rate on training points? **Zero.**
  In general, **training error** is an overly optimistic predictor of future performance.
- A better gauge: separate test set of 10,000 points.
  Test error = fraction of test points incorrectly classified.
- What test error would we expect for a random classifier? **90%.**
- Test error of nearest neighbor: **3.09%.**

Examples of errors:

Ideas for improvement: (1) k-NN (2) better distance function.

K-nearest neighbor classification

Classify a point using the labels of its k nearest neighbors among the training points.

MNIST:

k	1	3	5	7	9	11
Test error (%)	3.09	2.94	3.13	3.10	3.43	3.34

How to choose k in general?
Let $S \in \mathcal{Z}^n$ be the training set, where $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ is the space of *labeled* points. Let $\Gamma_k(S, x)$ be the prediction made on x using its k-NN in S.

1. Hold-out set. Choose a subset $V \subset S$ as a *validation set*.

$$\arg\min_k \sum_{(x,y)\in V} \mathbf{1}(\Gamma_k(S \setminus V, x) \neq y)$$

2. Leave-one-out cross-validation.

$$\arg\min_k \sum_{(x,y)\in S} \mathbf{1}(\Gamma_k(S \setminus \{(x,y)\}, x) \neq y)$$

Better distance functions

Let x be an image. Consider an image x' that is just like x, but is either:

- shifted one pixel to the right, or
- rotated slightly.

Then $\|x - x'\|$ could easily be quite large.

It makes sense to choose distance measures that are invariant under:

- Small translations and rotations. e.g. *tangent distance*.
- A broader family of natural deformations. e.g. *shape context*.

Test error rates:

ℓ_2	tangent distance	shape context
3.09	1.10	0.63

Are there families of distance functions that are often useful?

ℓ_p norms

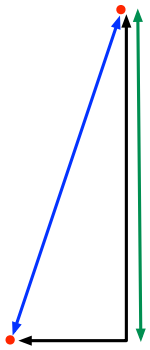
How can we measure the length of a vector in $\mathbb{R}^m$?
Usual choice is the *Euclidean norm*:

$$\|x\|_2 = \sqrt{\sum_{i=1}^m x_i^2}.$$

Generalization: For $p \geq 1$, the ℓ_p norm is

$$\|x\|_p = \left(\sum_{i=1}^m |x_i|^p\right)^{1/p}$$

- $p = 2$: Euclidean norm
- ℓ_1 norm: $\|x\|_1 = \sum_{i=1}^m |x_i|$
- ℓ_∞ norm: $\|x\|_\infty = \max_i |x_i|$



Metric spaces

A more general notion is a *metric space*.

Let $\mathcal{X}$ be the space in which data lie. A distance function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a *metric* if it satisfies these properties:

- $d(x, y) \geq 0$ (nonnegativity)
- $d(x, y) = 0$ if and only if $x = y$
- $d(x, y) = d(y, x)$ (symmetry)
- $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality)

For instance:

- $\mathcal{X} = \mathbb{R}^m$ and $d(x, y) = \|x - y\|_p$
- $\mathcal{X} = \{\text{strings over some alphabet}\}$ and $d = \text{edit distance}$.

Later in the course: methods for *learning* suitable distance measures.

Statistical learning theory

Model of reality: there is an (unknown) underlying distribution, call it P, from which pairs (x, y) are generated.

- Training points (x, y) come from this distribution.
- Future test points will also come from this distribution.

We want a classifier

$$f : \mathcal{X} \to \mathcal{Y}$$

which will do well on future data: in other words, do well on P. But we don't know P, so we treat the training data as a proxy for it.

The underlying distribution

Reality $\equiv$ the underlying distribution on pairs (x, y)

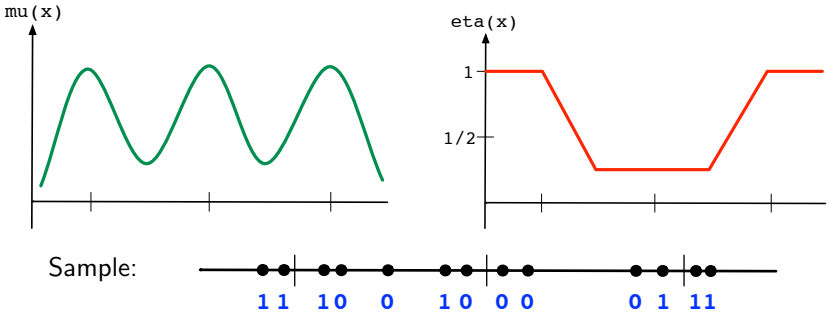
Can factor this distribution into two parts:

- A distribution on x. Call this μ.
- A distribution over labels y given x. In the binary case ($\mathcal{Y} = \{0, 1\}$) this can be specified as $\eta(x) = \Pr(Y = 1|x)$.

For instance, distribution of patients in a community:

x = age

$y = 1$ if visited doctor in the last year



Statistical learning theory, cont'd

Let's look at the binary case, $\mathcal{Y} = \{0, 1\}$.

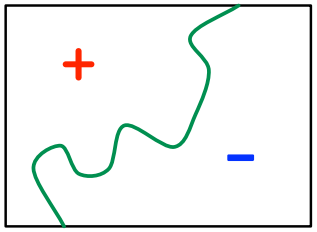
- There is an (unknown) underlying probability distribution on $\mathcal{X}$ from which *all* points are generated. Call this distribution μ.
- The label of any point x can, in general, be *stochastic*. It is a coin flip with bias $\eta(x) = \Pr(Y = 1|X = x)$.
- A classifier is a rule $h : \mathcal{X} \to \{0, 1\}$. Its misclassification rate, or *risk*, is $R(h) = \Pr(h(X) \neq Y)$.

The Bayes-optimal classifier

$$h^*(x) = \begin{cases} 1 & \text{if } \eta(x) > 1/2 \\ 0 & \text{otherwise} \end{cases}$$

has minimum risk,

$$R^* = R(h^*) = \mathbb{E}_X \min(\eta(X), 1 - \eta(X)).$$



Statistical theory of nearest neighbor

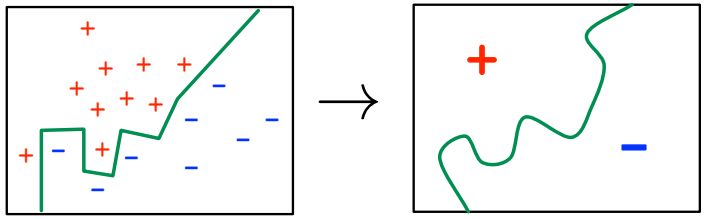
Let h_n be a classifier based on n training points from the underlying distribution.

- Its risk is $R(h_n) = \Pr(h_n(X) \neq Y)$.
- We say it is **consistent** if, as n grows to infinity, $R(h_n) \to R^*$.

Consistency of NN (Fix and Hodges, 1951; Cover and Hart, 1967):

- Suppose $\eta(x) = \Pr(Y = 1|X = x)$ is continuous in x.
- Set k to a growing function of n, with (1) $k \to \infty$ and (2) $k/n \to 0$, as $n \to \infty$.

Then k-NN is consistent.



Statistical theory of 1-NN

1-NN is not consistent.

E.g. $\mathcal{X} = \mathbb{R}$ and $\eta(x) \equiv 1/4$. Every label is a coin flip with bias $1/4$.

- Bayes optimal classifier: always predict 0. Risk $R^* = 1/4$.
- 1-NN risk: what is the probability that two coins of bias $1/4$ disagree?

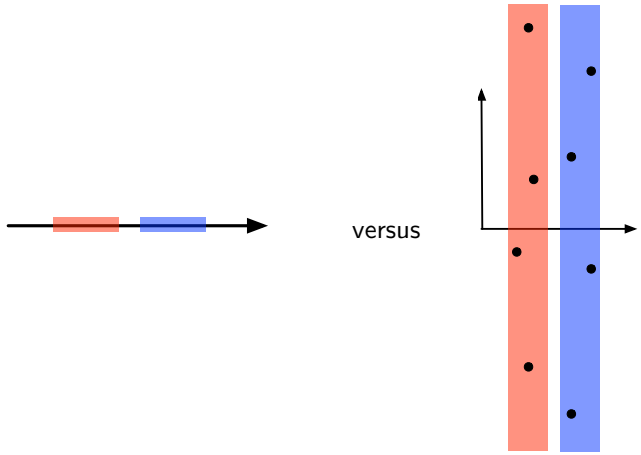
$$R(h_n) \;=\; 2 \cdot \frac{1}{4} \cdot \frac{3}{4} \;=\; \frac{3}{8} \;>\; \frac{1}{4}.$$

But (Cover-Hart 1967) it is always within a factor two of optimal:

$$R(h_n) \to 2R^*(1 - R^*).$$

Nearest neighbor: sensitivity to noise

Adding a single sufficiently noisy feature can wreak havoc with the classifier.



Solutions: feature selection/reweighting; distance function learning.

Fast NN search

Naive search is $O(n)$ for training set of size n: very slow.

Two popular approaches to fast nearest neighbor search, for data set $S \subset \mathcal{X}$ and query q.

1. Locality sensitive hashing
   Collection of special hash functions $h_1, \ldots, h_m : \mathcal{X} \to \mathbb{Z}$.
   Search for nearest neighbor in

$$\bigcup_{i=1}^{m} \{x \in S : h_i(x) = h_i(q)\}$$

   This set is smaller than S, and is likely to contain the nearest neighbor of q.
2. Tree-based search
   Build tree structure on S, and use it to discard subsets of S that are far from a query q.
   Common options: k-d tree, PCA tree, cover tree.

K-d trees for NN search

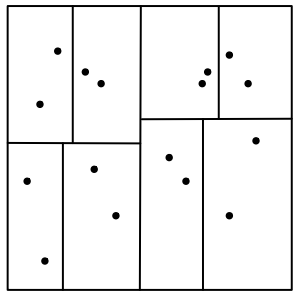
A hierarchical, rectilinear spatial partition.

For data set $S \subset \mathbb{R}^p$:

- Pick a coordinate $1 \leq i \leq p$.
- Compute $v = \text{median}(\{x_i : x \in S\})$.
- Split S into two halves:

$$S_L = \{x \in S : x_i < v\}$$
$$S_R = \{x \in S : x_i \geq v\}$$

- Recurse on S_L, S_R



Two types of search, given a query $q \in \mathbb{R}^p$:

- *Defeatist search*: Route q to a leaf cell and return the NN in that cell. This might not be the true NN.
- *Comprehensive search*: Grow the search region to other cells that cannot be ruled out using the triangle inequality.

Decision trees

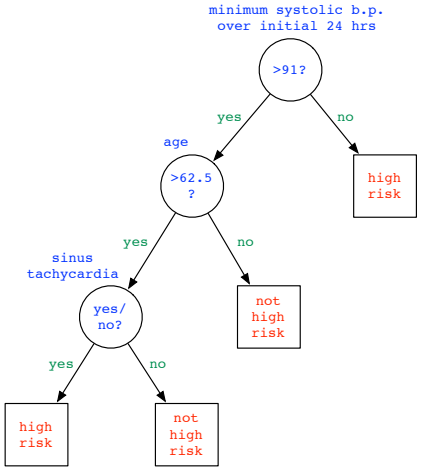
CSE 250B

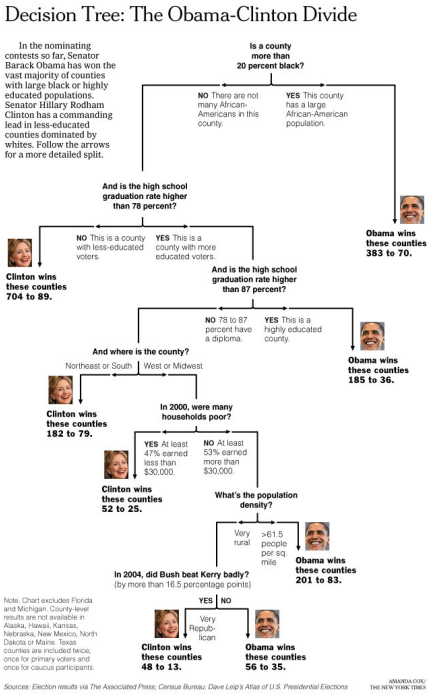
Decision trees

Study at UCSD Medical Center, late 1970s.

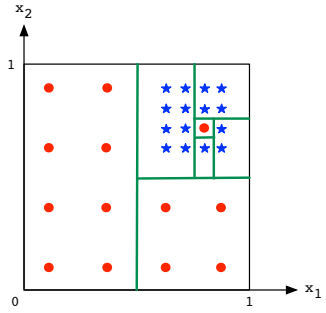
Goal: identify patients at risk of dying within 30 days after heart attack.

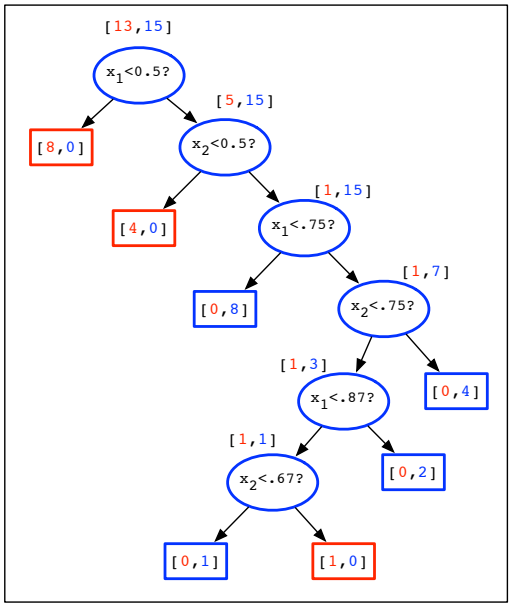
Data set: 215 patients, 37 (=20%) died. 19 relevant variables.





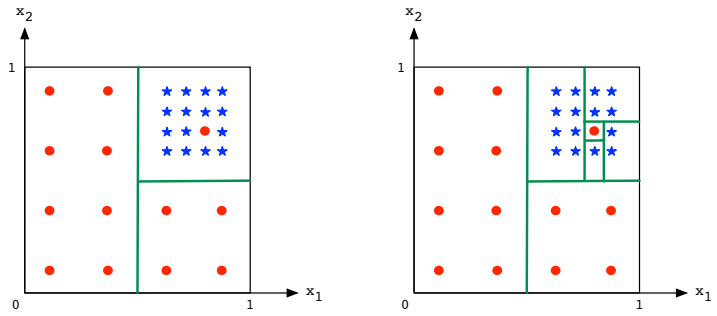
Example: building a decision tree





Overfitting?

Go back a few steps...



The final partition does better on the training data, but is more complex.
That one point might have been an outlier anyway.

We have probably ended up **overfitting** the data.

Building a decision tree

Greedy algorithm: build tree top-down.

- Start with a single node containing all data points
- Repeat:
  - Look at all current leaves and all possible splits
  - Choose the split that most decreases the uncertainty in prediction

We need a measure of **uncertainty in prediction**.

Decision tree issues

A very expressive family of classifiers:

- Can accommodate any type of data: real, Boolean, categorical, ...
- Can accommodate any number of classes
- Can fit any data set
- Statistically consistent

But this also means that there is serious danger of overfitting.

Uncertainty in prediction

Say there are two labels:

$+$ p fraction of the points
$-$ $1-p$ fraction of the points

What uncertainty score should we give to this?

1. Misclassification rate

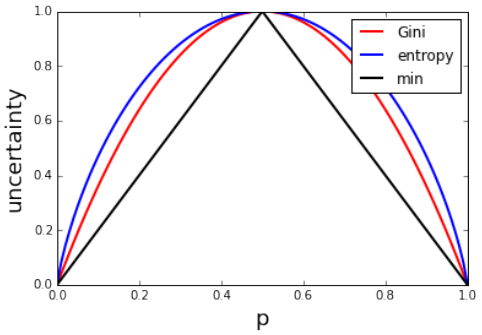
   $$\min\{p, 1-p\}$$

2. Gini index

   $$2p(1-p)$$

3. Entropy

   $$p \log \frac{1}{p} + (1-p) \log \frac{1}{1-p}$$



Uncertainty: k classes

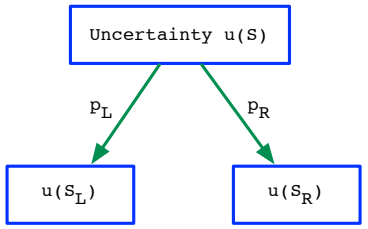
Suppose there are k classes, with probabilities $p_1, p_2, \ldots, p_k$.

	$k = 2$	General k
Misclassification rate	$\min\{p, 1-p\}$	$1 - \max_i p_i = 1 - \|p\|_\infty$
Gini index	$2p(1-p)$	$\sum_{i \neq j} p_i p_j = 1 - \|p\|^2$
Entropy	$p \log \frac{1}{p} + (1-p) \log \frac{1}{1-p}$	$\sum_i p_i \log \frac{1}{p_i}$

Benefit of a split

Let $u(S)$ be the uncertainty score for a set of labeled points S.

Consider a particular split:



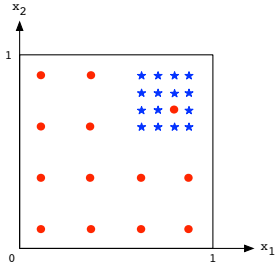
Of the points in S:

- p_L fraction go to S_L
- p_R fraction go to S_R

Benefit of split = reduction in uncertainty:

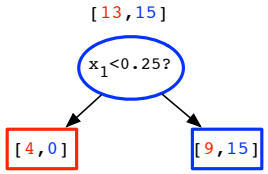
$$\left(u(S) - \underbrace{(p_L\, u(S_L) + p_R\, u(S_R))}_{\text{expected uncertainty after split}} \right) \times |S|$$

Benefit of a split: example

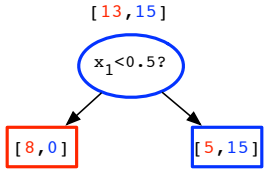


Initial uncertainty (Gini):

$$2 \times \frac{13}{28} \times \frac{15}{28}$$



$$p_L u_L + p_R u_R = \frac{4}{28} \cdot 0 + \frac{24}{28} \cdot 2 \cdot \frac{9}{24} \cdot \frac{15}{24} = \frac{45}{112}$$



$$p_L u_L + p_R u_R = \frac{8}{28} \cdot 0 + \frac{20}{28} \cdot 2 \cdot \frac{5}{20} \cdot \frac{15}{20} = \frac{30}{112}$$

Building a decision tree

- Start with a single node containing all data points
- Repeat:
  - Look at all current leaves and all possible splits
  - Choose the split with the greatest benefit

When to stop?

- When each leaf is pure?
- When the tree is already pretty big?
- When each leaf has uncertainty below some threshold?

Common strategy: keep going until leaves are pure.
Then, shorten the tree by **pruning**, to correct for overfitting.

What is overfitting?

Data comes from an unknown, underlying distribution D on $\mathcal{X} \times \mathcal{Y}$.
All we ever see are samples from D.

For a data set $(x_1, y_1), \ldots, (x_n, y_n)$, the **training error** of a classifier $h : \mathcal{X} \to \mathcal{Y}$ is

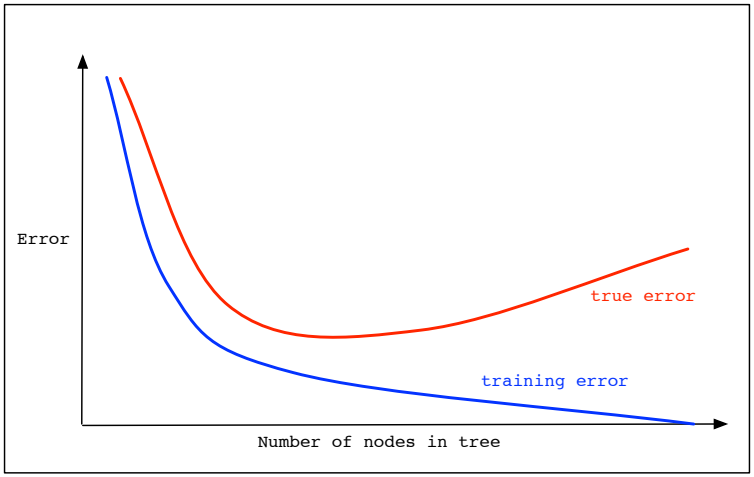
$$\widehat{\text{err}}(h) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(h(x_i) \neq y_i).$$

What we really care about is the **true error** of h with respect to D:

$$\text{err}(h) = \Pr_{(x,y)\sim D}(h(x) \neq y).$$

How are these two quantities related?

Overfitting: picture

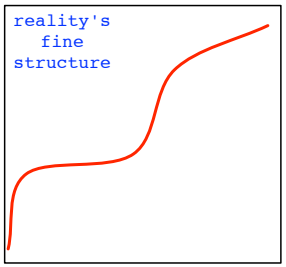


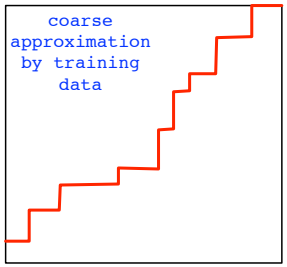
As we make our tree more and more complicated:

- training error keeps going down
- but, at some point, true error starts increasing!

Overfitting: perspectives

- The true underlying distribution D is the one whose structure we would like to capture.
- The training data reflects the structure of D, so it helps us.
- But it also has chance structure of its own – we must avoid modeling this.





Another perspective: it is absurd to fit a line to a point.
More generally, it is not good to use a model that is so complex that there isn't enough data to reliably estimate its parameters.

Decision tree construction and pruning

1. Split the training set into two parts
   - A smaller training set S
   - A validation set V (surrogate test set, model of reality)
2. Build a full decision tree using S
3. Then prune using V
   Use dynamic programming to find the pruning that does best on V

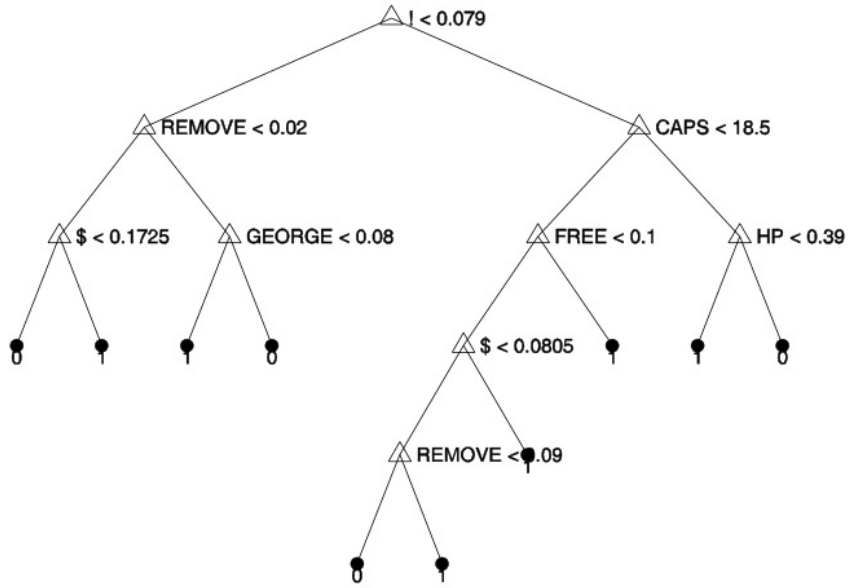
Of course, V can have chance structure too — but its chance structure is unlikely to coincide with that of S.

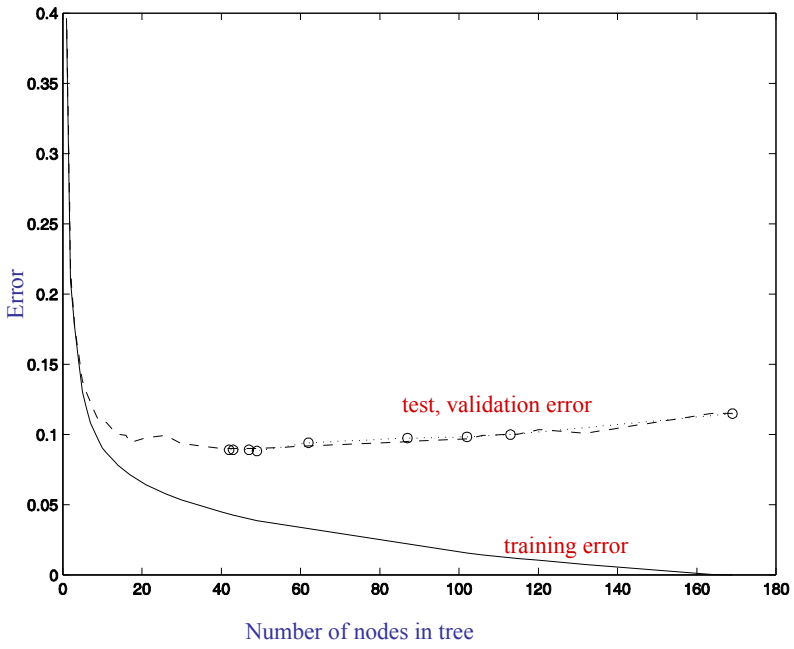
SPAMbase data set

- 4601 email messages, labeled SPAM or NOT SPAM.
- 39.4% are SPAM.
- Each email is represented by 57 features.
  - 48 check for specific words, e.g. FREE
  - 6 check for specific characters, e.g. !
  - 3 others, e.g. longest run of capitals

Randomly divide into three parts:

50% training data
25% validation set
25% test set





How accurate is the validation error?

For any classifier h and underlying distribution D on $\mathcal{X} \times \mathcal{Y}$:

true error $\quad \text{err}(h) = \Pr_{(x,y)\sim D}(h(x) \neq y)$

error on set A $\quad \text{err}(h, A) = \frac{1}{|A|} \sum_{(x,y)\in A} \mathbf{1}(h(x) \neq y)$

Suppose A is chosen i.i.d. (independent, identically distributed) from D. Then (over the random choice of A),

$$\mathbb{E}[\text{err}(h, A)] = \text{err}(h)$$

and the standard deviation of $\text{err}(h, A)$ is roughly $1/\sqrt{|A|}$.

Caution! In this scenario:

- Set A is used to assess a single, prespecified classifier h.
- If h was created using A as a training set, the result does not apply. In such situations, $\text{err}(h, A)$ could be a very poor estimate of $\text{err}(h)$.

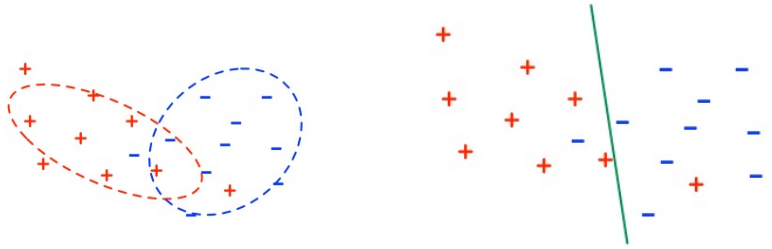
Classification with generative models

CSE 250B

Classification with parametrized models

Classifiers with a fixed number of parameters can represent a limited set of functions. Learning a model is about picking a good approximation.

Typically the x's are points in p-dimensional Euclidean space, $\mathbb{R}^p$.



Two ways to classify:

- **Generative**: model the individual classes.
- **Discriminative**: model the decision boundary between the classes.

Quick review of conditional probability

Formula for conditional probability: for any events A, B,

$$\Pr(A|B) = \frac{\Pr(A \cap B)}{\Pr(B)}.$$

Applied twice, this yields Bayes' rule:

$$\Pr(H|E) = \frac{\Pr(E|H)}{\Pr(E)}\Pr(H).$$

Example: Toss ten coins. What is the probability that the first is heads, given that nine of them are heads?

H = first coin is heads

E = nine of the ten coins are heads

$$\Pr(H|E) = \frac{\Pr(E|H)}{\Pr(E)} \cdot \Pr(H) = \frac{\binom{9}{8}\frac{1}{2^9}}{\binom{10}{9}\frac{1}{2^{10}}} \cdot \frac{1}{2} = \frac{9}{10}$$

Summation rule

Suppose events $A_1, \ldots, A_k$ are disjoint events, one of which must occur. Then for any other event E,

$$\begin{aligned}\Pr(E) &= \Pr(E, A_1) + \Pr(E, A_2) + \cdots + \Pr(E, A_k) \\ &= \Pr(E|A_1)\Pr(A_1) + \Pr(E|A_2)\Pr(A_2) + \cdots + \Pr(E|A_k)\Pr(A_k)\end{aligned}$$

Example: Sex bias in graduate admissions. In 1969, there were 12673 applicants for graduate study at Berkeley. 44% of the male applicants were accepted, and 35% of the female applicants.

Over the sample space of applicants, define:

M = male

F = female

A = admitted

So: $\Pr(A|M) = 0.44$ and $\Pr(A|F) = 0.35$.

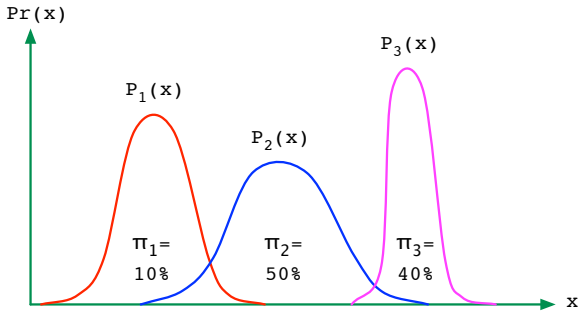
In every department, the accept rate for female applicants was at least as high as the accept rate for male applicants. How could this be?

Generative models

An unknown underlying distribution D over $\mathcal{X} \times \mathcal{Y}$.
Generating a point (x, y) in two steps:

1. Last week: first choose x, then choose y given x.
2. Now: first choose y, then choose x given y.

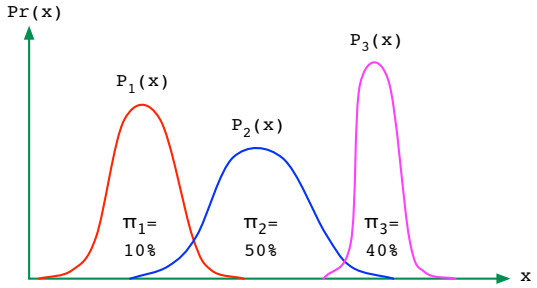
Example:
$\mathcal{X} = \mathbb{R}$
$\mathcal{Y} = \{1, 2, 3\}$



The overall density is a mixture of the individual densities,

$$\Pr(x) = \pi_1 P_1(x) + \cdots + \pi_k P_k(x).$$

The Bayes-optimal prediction



Labels $\mathcal{Y} = \{1, 2, \ldots, k\}$, density $\Pr(x) = \pi_1 P_1(x) + \cdots + \pi_k P_k(x)$.

For any $x \in \mathcal{X}$ and any label j,

$$\Pr(y = j|x) = \frac{\Pr(y = j)\Pr(x|y = j)}{\Pr(x)} = \frac{\pi_j P_j(x)}{\sum_{i=1}^{k} \pi_i P_i(x)}$$

Bayes-optimal prediction: $h^*(x) = \arg\max_j \pi_j P_j(x)$.

Estimating the π_j is easy. Estimating the P_j is hard.

Estimating class-conditional distributions

Estimating an arbitrary distribution in $\mathbb{R}^p$:

- Can be done, e.g. with kernel density estimation.
- But number of samples needed is exponential in p.

Instead: approximate each P_j with a simple, parametric distribution.

Some options:

- Product distributions.
  Assume coordinates are independent: naive Bayes.
- Multivariate Gaussians.
  Linear and quadratic discriminant analysis.
- More general graphical models.

Naive Bayes

Labels $\mathcal{Y} = \{1, 2, \ldots, k\}$, density $\Pr(x) = \pi_1 P_1(x) + \cdots + \pi_k P_k(x)$.

Binarized MNIST:

- $k = 10$ classes
- $\mathcal{X} = \{0, 1\}^{784}$

Assume that **within each class**, the individual pixel values are independent:

$$P_j(x) = P_{j1}(x_1) \cdot P_{j2}(x_2) \cdots P_{j,784}(x_{784}).$$

Each P_{ji} is a coin flip: trivial to estimate!

Smoothed estimate of coin bias

Pick a class j and a pixel i. We need to estimate

$$p_{ji} = \Pr(x_i = 1 | y = j).$$

Out of a training set of size n,

$$n_j = \#\text{ of instances of class } j$$
$$n_{ji} = \#\text{ of instances of class } j \text{ with } x_i = 1$$

Then the maximum-likelihood estimate of p_{ji} is

$$\widehat{p}_{ji} = n_{ji}/n_j.$$

This causes problems if $n_{ji} = 0$. Instead, use "Laplace smoothing":

$$\widehat{p}_{ji} = \frac{n_{ji}+1}{n_j+2}.$$

Form of the classifier

Data space $\mathcal{X} = \{0,1\}^p$, label space $\mathcal{Y} = \{1,\ldots,k\}$. Estimate:

- $\{\pi_j : 1 \le j \le k\}$
- $\{p_{ji} : 1 \le j \le k, 1 \le i \le p\}$

Then classify point x as

$$\arg\max_j \quad \pi_j \prod_{i=1}^{p} p_{ji}^{x_i}(1-p_{ji})^{1-x_i}.$$

To avoid underflow: take the log:

$$\arg\max_j \quad \underbrace{\log \pi_j + \sum_{i=1}^{p} (x_i \log p_{ji} + (1-x_i)\log(1-p_{ji}))}_{\text{of the form } w \cdot x + b}$$

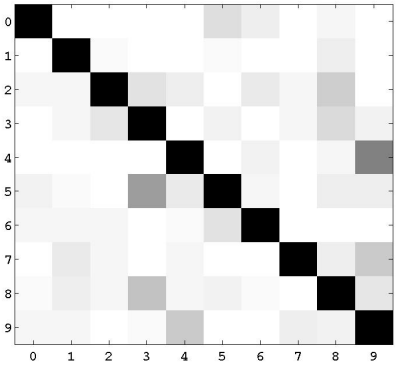
A linear classifier!

Example: MNIST

Result of training: mean vectors for each class.

Test error rate: 15.54%.

Visualization of the "confusion matrix" $\longrightarrow$



Other types of data

How would you handle data:

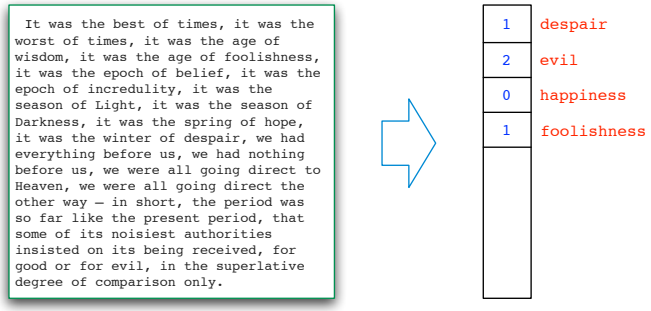
- Whose features take on more than two discrete values (such as ten possible colors)?
- Whose features are real-valued?
- Whose features are positive integers?
- Whose features are mixed: some real, some Boolean, etc?

How would you handle "missing data": situations in which data points occasionally (or regularly) have missing entries?

- At train time: **???**
- At test time: **???**

Handling text data

Bag-of-words: vectorial representation of text documents.



- Fix V = some vocabulary.
- Treat each document as a vector of length $|V|$:

$$x = (x_1, x_2, \ldots, x_{|V|}),$$

where x_i = # of times the ith word appears in the document.

A standard distribution over such document-vectors x: the **multinomial**.

Multinomial naive Bayes

Multinomial distribution over a vocabulary V:

$$p = (p_1, \ldots, p_{|V|}), \quad \text{such that} \quad p_i \geq 0 \text{ and } \sum_i p_i = 1$$

Document $x = (x_1, \ldots, x_{|V|})$ has probability $p_1^{x_1} p_2^{x_2} \cdots p_{|V|}^{x_{|V|}}$.

For naive Bayes: one multinomial distribution per class.

- Class probabilities $\pi_1, \ldots, \pi_k$
- Multinomials $p^{(1)} = (p_{11}, \ldots, p_{1|V|}), \ldots, p^{(k)} = (p_{k1}, \ldots, p_{k|V|})$

Classify document x as

$$\arg\max_j \quad \pi_j \prod_{i=1}^{|V|} p_{ji}^{x_i}.$$

(As always, take log to avoid underflow: linear classifier.)

Improving performance of multinomial naive Bayes

A variety of heuristics that are standard in text retrieval, such as:

1. Compensating for burstiness.
   Problem: Once a word has appeared in a document, it has a much higher chance of appearing again.

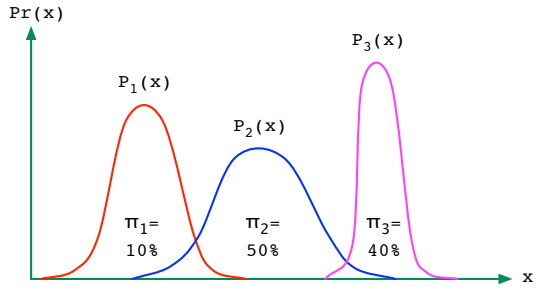
   Solution: Instead of the number of occurrences f of a word, use $\log(1 + f)$.

2. Downweighting common words.
   Problem: Common words can have a unduly large influence on classification.

   Solution: Weight each word w by **inverse document frequency**:

$$\log \frac{\#\text{ docs}}{\#(\text{docs containing } w)}$$

Recall: generative model framework



Labels $\mathcal{Y} = \{1, 2, \ldots, k\}$, density $\Pr(x) = \pi_1 P_1(x) + \cdots + \pi_k P_k(x)$.

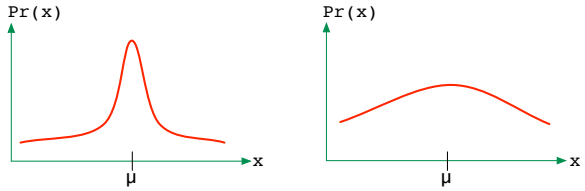
Approximate each P_j with a simple, parametric distribution:

- Product distributions.
  Assume coordinates are independent: naive Bayes.
- Multivariate Gaussians.
  Linear and quadratic discriminant analysis.
- More general graphical models.

Variance

If you had to summarize the entire distribution of a r.v. X by a single number, you would use the mean (or median). Call it μ.

But these don't capture the *spread* of X:



What would be a good measure of spread? How about the average distance away from the mean: $\mathbb{E}(|X-\mu|)$?

For convenience, take the square instead of the absolute value.

Variance: $\quad \text{var}(X) = \mathbb{E}(X-\mu)^2 = \mathbb{E}(X^2)-\mu^2,$

where $\mu = \mathbb{E}(X)$. The variance is always ≥ 0.

Variance: example

Recall: $\text{var}(X) = \mathbb{E}(X-\mu)^2 = \mathbb{E}(X^2)-\mu^2$, where $\mu = \mathbb{E}(X)$.

Toss a coin of bias p. Let $X \in \{0,1\}$ be the outcome.

$$\begin{aligned}\mathbb{E}(X) &= p\\ \mathbb{E}(X^2) &= p\\ \mathbb{E}(X-\mu)^2 &= p^2\cdot(1-p)+(1-p)^2\cdot p = p(1-p)\\ \mathbb{E}(X^2)-\mu^2 &= p-p^2 = p(1-p)\end{aligned}$$

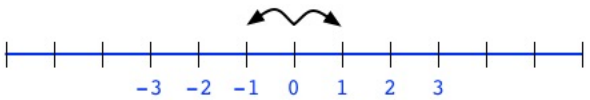
This variance is highest when $p = 1/2$ (fair coin).

The standard deviation of X is $\text{std}(X) = \sqrt{\text{var}(X)}$.
It is the average amount by which X differs from its mean.

Variance of a sum

$\text{var}(X_1+\cdots+X_k) = \text{var}(X_1)+\cdots+\text{var}(X_k)$ if the X_i are independent.

Symmetric random walk. A drunken man sets out from a bar. At each time step, he either moves one step to the right or one step to the left, with equal probabilities. Roughly where is he after n steps?



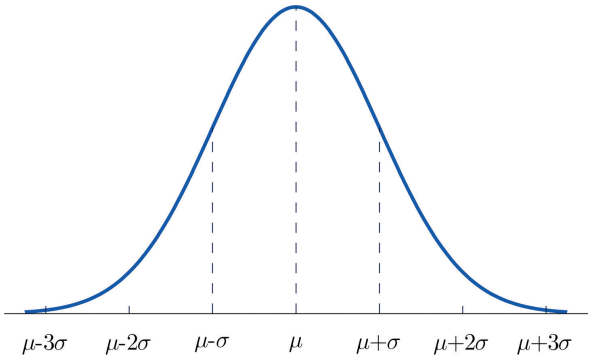
Let $X_i \in \{-1,1\}$ be his ith step. Then $\mathbb{E}(X_i) = ?0$ and $\text{var}(X_i) = ?1$.

His position after n steps is $X = X_1+\cdots+X_n$.

$$\begin{aligned}\mathbb{E}(X) &= 0\\ \text{var}(X) &= n\\ \text{stddev}(X) &= \sqrt{n}\end{aligned}$$

He is likely to be pretty close to where he started!

The univariate Gaussian



The Gaussian $N(\mu,\sigma^2)$ has mean μ, variance σ^2, and density function

$$p(x) = \frac{1}{(2\pi\sigma^2)^{1/2}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right).$$

But what if we have **two** variables?

Bivariate distributions

Simplest option: treat each variable as independent.

Example: For a large collection of people, measure the two variables

$$H = \text{height}$$
$$W = \text{weight}$$

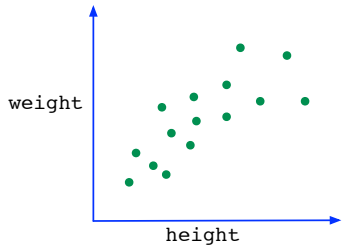
Independence would mean

$$\Pr(H = h, W = w) \;=\; \Pr(H = h)\Pr(W = w),$$

which would also imply $\mathbb{E}(HW) = \mathbb{E}(H)\mathbb{E}(W)$.

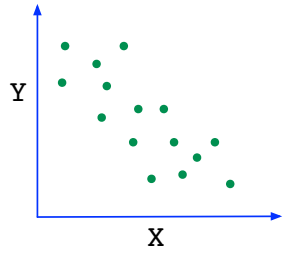
Is this an accurate approximation?
No: we'd expect height and weight to be **positively correlated**.

Types of correlation

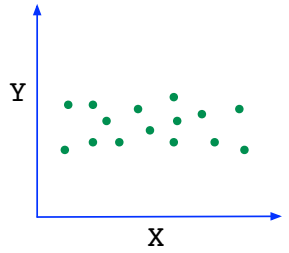


H, W positively correlated.
This also implies

$$\mathbb{E}(HW) > \mathbb{E}(H)\mathbb{E}(W).$$

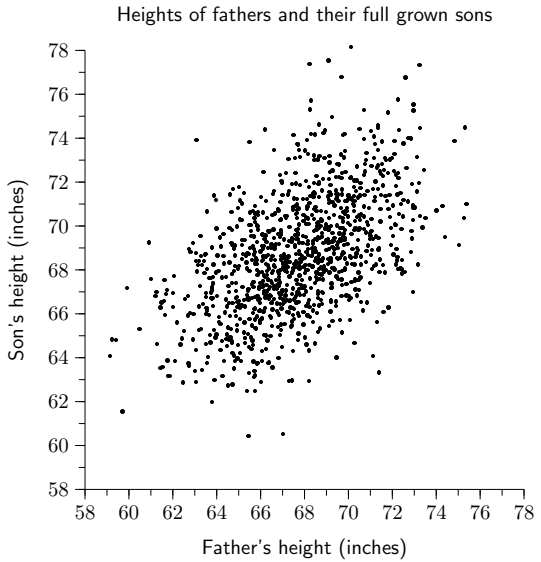


X, Y negatively correlated



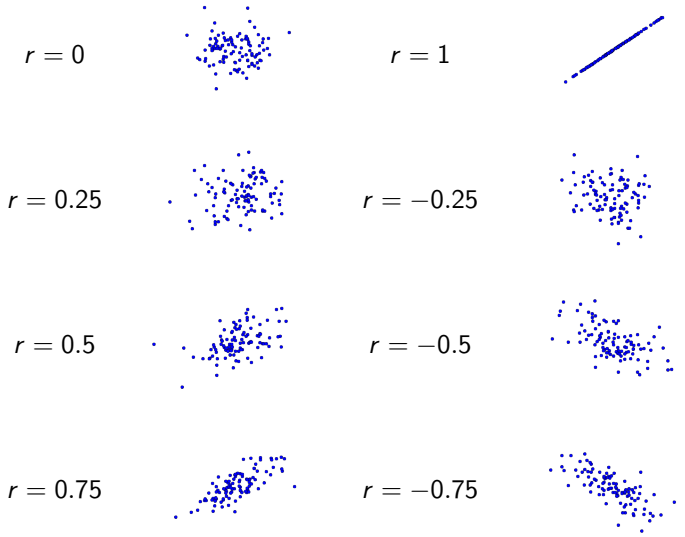
X, Y uncorrelated

Pearson (1903): fathers and sons



How to quantify the degree of correlation?

Correlation pictures



Covariance and correlation

Suppose X has mean μ_X and Y has mean μ_Y.

- Covariance

$$\text{cov}(X, Y) = \mathbb{E}[(X - \mu_X)(Y - \mu_Y)] = \mathbb{E}[XY] - \mu_X\mu_Y$$

  Maximized when $X = Y$, in which case it is $\text{var}(X)$.
  In general, it is at most $\text{std}(X)\text{std}(Y)$.

- Correlation

$$\text{corr}(X, Y) = \frac{\text{cov}(X, Y)}{\text{std}(X)\text{std}(Y)}$$

  This is always in the range $[-1, 1]$.

Covariance and correlation: example 1

$$\text{cov}(X, Y) = \mathbb{E}[(X - \mu_X)(Y - \mu_Y)] \;=\; \mathbb{E}[XY] - \mu_X\mu_Y$$
$$\text{corr}(X, Y) = \frac{\text{cov}(X, Y)}{\text{std}(X)\text{std}(Y)}$$

x	y	$\Pr(x, y)$
-1	-1	$1/3$
-1	1	$1/6$
1	-1	$1/3$
1	1	$1/6$

$$\mu_X = 0$$
$$\mu_Y = -1/3$$
$$\text{var}(X) = 1$$
$$\text{var}(Y) = 8/9$$
$$\text{cov}(X, Y) = 0$$
$$\text{corr}(X, Y) = 0$$

In this case, X, Y are independent. Independent variables always have zero covariance and correlation.

Covariance and correlation: example 2

$$\text{cov}(X, Y) = \mathbb{E}[(X - \mu_X)(Y - \mu_Y)] \;=\; \mathbb{E}[XY] - \mu_X\mu_Y$$
$$\text{corr}(X, Y) = \frac{\text{cov}(X, Y)}{\text{std}(X)\text{std}(Y)}$$

x	y	$\Pr(x, y)$
-1	-10	$1/6$
-1	10	$1/3$
1	-10	$1/3$
1	10	$1/6$

$$\mu_X = 0$$
$$\mu_Y = 0$$
$$\text{var}(X) = 1$$
$$\text{var}(Y) = 100$$
$$\text{cov}(X, Y) = -10/3$$
$$\text{corr}(X, Y) = -1/3$$

In this case, X and Y are negatively correlated.

The bivariate (2-d) Gaussian

A distribution over $(x, y) \in \mathbb{R}^2$, parametrized by:

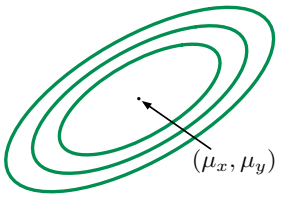
- **Mean** $(\mu_x, \mu_y) \in \mathbb{R}^2$
- **Covariance matrix**

$$\Sigma = \begin{bmatrix} \Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_{yy} \end{bmatrix}$$

  where $\Sigma_{xx} = \text{var}(X)$, $\Sigma_{yy} = \text{var}(Y)$, $\Sigma_{xy} = \Sigma_{yx} = \text{cov}(X, Y)$

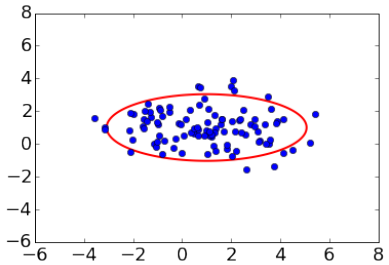
$$\text{Density} \quad p(x, y) = \frac{1}{2\pi|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}\begin{bmatrix} x - \mu_x \\ y - \mu_y \end{bmatrix}^T \Sigma^{-1} \begin{bmatrix} x - \mu_x \\ y - \mu_y \end{bmatrix}\right)$$

The density is highest at the mean, and falls off in ellipsoidal contours.

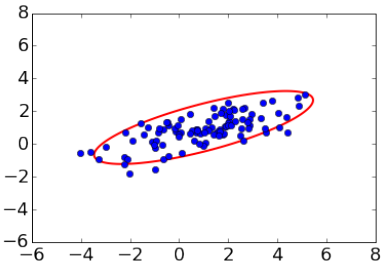


Bivariate Gaussian: examples

In either case, the mean is $(1, 1)$.

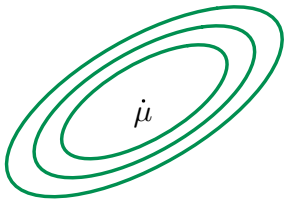


$$\Sigma = \begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\Sigma = \begin{bmatrix} 4 & 1.5 \\ 1.5 & 1 \end{bmatrix}$$

The multivariate Gaussian



$N(\mu, \Sigma)$: Gaussian in $\mathbb{R}^p$

- mean: $\mu \in \mathbb{R}^p$
- covariance: $p \times p$ matrix Σ

Density $p(x) = \dfrac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} \exp\left(-\dfrac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$

Let $X = (X_1, X_2, \ldots, X_p)$ be a random draw from $N(\mu, \Sigma)$.

- μ is the vector of coordinate-wise means:

$$\mu_1 = \mathbb{E}X_1, \ \mu_2 = \mathbb{E}X_2, \ldots, \ \mu_p = \mathbb{E}X_p.$$

- Σ is a matrix containing all pairwise covariances:

$$\Sigma_{ij} = \Sigma_{ji} = \text{cov}(X_i, X_j) \quad \text{if } i \neq j$$
$$\Sigma_{ii} = \text{var}(X_i)$$

- In matrix/vector form: $\mu = \mathbb{E}X$ and $\Sigma = \mathbb{E}(X-\mu)(X-\mu)^T$.

Special case: spherical Gaussian

The X_i are independent and all have the same variance σ^2. Thus

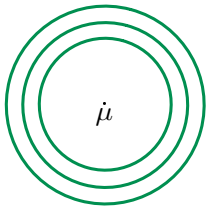
$$\Sigma = \sigma^2 I_p = \text{diag}(\sigma^2, \sigma^2, \ldots, \sigma^2)$$

(off-diagonal elements zero, diagonal elements σ^2).

Each X_i is an independent univariate Gaussian $N(\mu_i, \sigma^2)$:

$$\Pr(x) = \prod_{i=1}^{p}\left(\frac{1}{\sigma\sqrt{2\pi}}e^{-(x_i-\mu_i)^2/2\sigma^2}\right) = \frac{1}{(2\pi)^{p/2}\sigma^p}\exp\left(-\frac{\|x-\mu\|^2}{2\sigma^2}\right)$$

Density at a point depends only on its distance from μ:



Special case: diagonal Gaussian

The X_i are independent, with variances σ_i^2. Thus

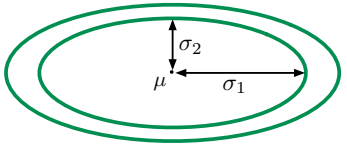
$$\Sigma = \text{diag}(\sigma_1^2, \ldots, \sigma_p^2)$$

(all off-diagonal elements zero).

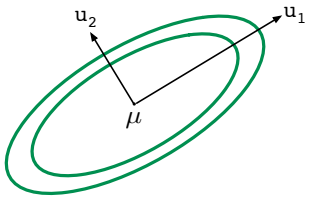
Each X_i is an independent univariate Gaussian $N(\mu_i, \sigma_i^2)$:

$$p(x) = \frac{1}{(2\pi)^{p/2}\sigma_1\cdots\sigma_p}\exp\left(-\sum_{i=1}^{p}\frac{(x_i-\mu_i)^2}{2\sigma_i^2}\right)$$

Contours of equal density are axis-aligned ellipsoids centered at μ:



The general Gaussian $N(\mu, \Sigma)$ in $\mathbb{R}^p$



Eigendecomposition of Σ yields:

- **Eigenvalues** $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p$
- Corresponding **eigenvectors** $u_1, \ldots, u_p$

Recall density: $$p(x) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}\underbrace{(x-\mu)^T\Sigma^{-1}(x-\mu)}_{\text{What is this?}}\right)$$

If we write $S = \Sigma^{-1}$ then S is a $p \times p$ matrix and

$$(x-\mu)^T\Sigma^{-1}(x-\mu) = \sum_{i,j} S_{ij}(x_i-\mu_i)(x_j-\mu_j),$$

a **quadratic function** of x.

Binary classification with Gaussian generative model

Estimate class probabilities π_1, π_2 and fit a Gaussian to each class:

$$P_1 = N(\mu_1, \Sigma_1), P_2 = N(\mu_2, \Sigma_2)$$

E.g. If data points $x^{(1)}, \ldots, x^{(m)} \in \mathbb{R}^p$ are class 1:

$$\mu_1 = \frac{1}{m}\left(x^{(1)} + \cdots + x^{(m)}\right) \quad \text{and} \quad \Sigma_1 = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)}-\mu_1)(x^{(i)}-\mu_1)^T$$

Given a new point x, predict class 1 iff:

$$\pi_1 P_1(x) > \pi_2 P_2(x) \quad \Leftrightarrow \quad x^T M x + 2w^T x \geq \theta,$$

where:

$$M = \frac{1}{2}(\Sigma_2^{-1} - \Sigma_1^{-1})$$
$$w = \Sigma_1^{-1}\mu_1 - \Sigma_2^{-1}\mu_2$$

and θ is a constant depending on the various parameters.

$\Sigma_1 = \Sigma_2$: linear decision boundary. Otherwise, quadratic boundary.

Linear decision boundary

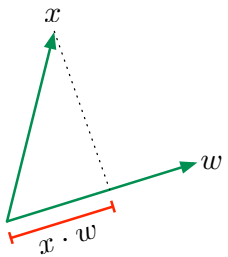
When $\Sigma_1 = \Sigma_2 = \Sigma$: choose class 1 iff

$$x \cdot \underbrace{\Sigma^{-1}(\mu_1 - \mu_2)}_{w} \geq \theta.$$

What does $x \cdot w$ (or equivalently $x^T w$, or $w^T x$) mean?

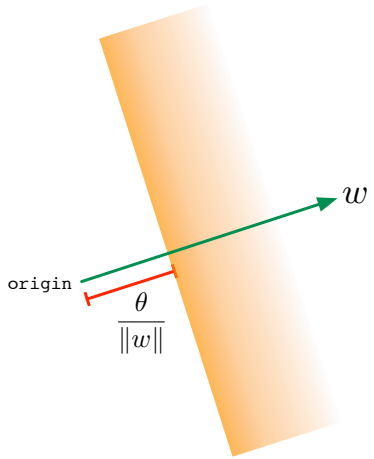
Algebraically: $x \cdot w = w \cdot x = x^T w = w^T x = \sum_{i=1}^{p} x_i w_i$

Geometrically: Suppose w is a unit vector (that is, $\|w\| = 1$). Then $x \cdot w$ is the projection of vector x onto direction w.



Linear decision boundary

Let w be any vector in $\mathbb{R}^p$. What is meant by decision rule $w \cdot x \geq \theta$?

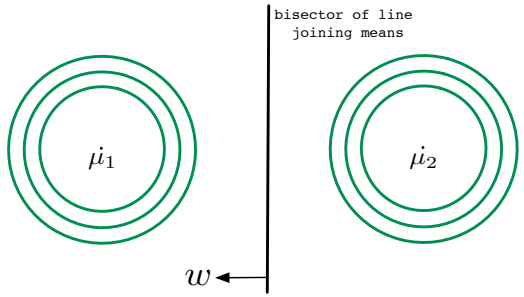


Common covariance: $\Sigma_1 = \Sigma_2 = \Sigma$

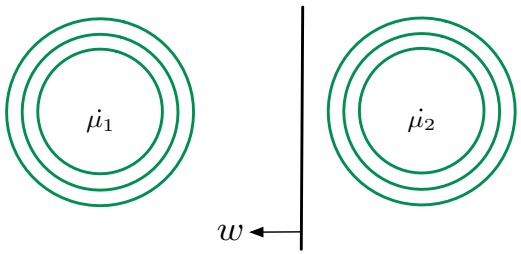
Linear decision boundary: choose class 1 iff

$$x \cdot \underbrace{\Sigma^{-1}(\mu_1 - \mu_2)}_{w} \geq \theta.$$

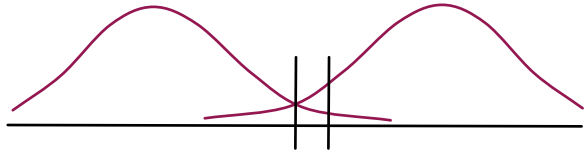
Example 1: Spherical Gaussians with $\Sigma = I_p$ and $\pi_1 = \pi_2$.



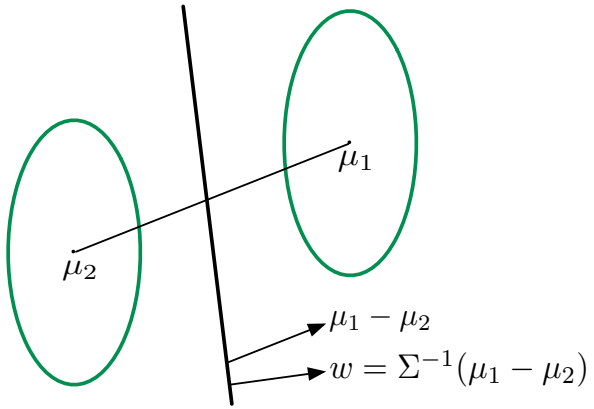
Example 2: Again spherical, but now $\pi_1 > \pi_2$.



One-d projection onto w:



Example 3: Non-spherical.



Rule: $w \cdot x \geq \theta$

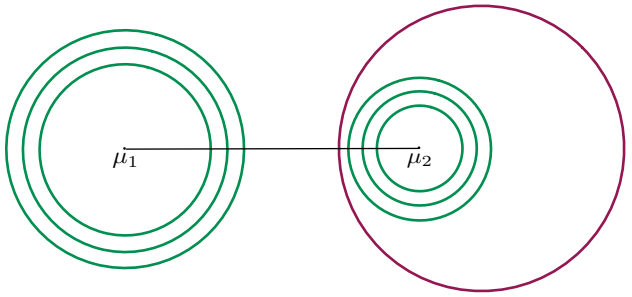
- w, θ dictated by probability model, assuming it is a perfect fit
- Common practice: choose w as above, but fit θ to minimize training/validation error

Different covariances: $\Sigma_1 \neq \Sigma_2$

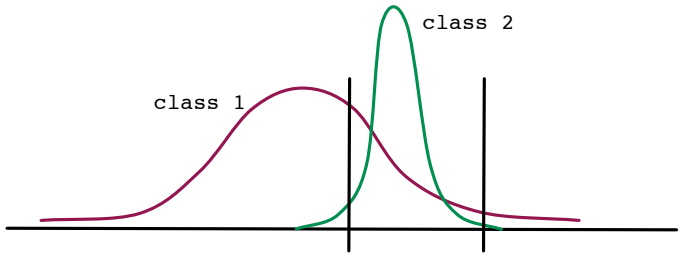
Quadratic boundary: choose class 1 iff $x^T M x + 2w^T x \geq \theta$, where:

$$M = \frac{1}{2}(\Sigma_2^{-1} - \Sigma_1^{-1})$$
$$w = \Sigma_1^{-1}\mu_1 - \Sigma_2^{-1}\mu_2$$

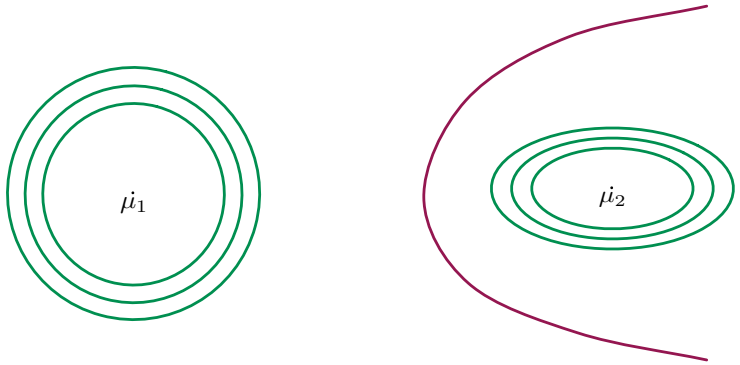
Example 1: $\Sigma_1 = \sigma_1^2 I_p$ and $\Sigma_2 = \sigma_2^2 I_p$ with $\sigma_1 > \sigma_2$



Example 2: Same thing in 1-d. $\mathcal{X} = \mathbb{R}$.



Example 3: A parabolic boundary.



Many other possibilities!

Multiclass discriminant analysis

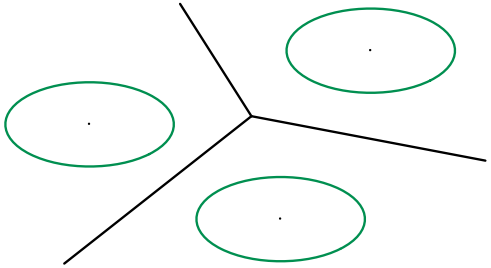
k classes: weights π_j, class-conditional distributions $P_j = N(\mu_j, \Sigma_j)$.

Each class has an associated **quadratic** function

$$f_j(x) = \log(\pi_j P_j(x))$$

To class a point x, pick $\arg\max_j f_j(x)$.

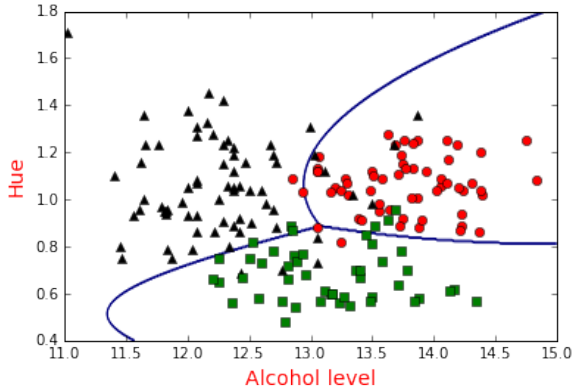
If $\Sigma_1 = \cdots = \Sigma_k$, the boundaries are **linear**.



Example: "wine" data set

Data from three wineries from the same region of Italy

- 13 attributes: hue, color intensity, flavanoids, ash content, ...
- 178 instances in all: split into 118 train, 60 test



Test error using multiclass discriminant analysis: 1/60

Example: MNIST

To each digit, fit:

- class probability π_j
- mean $\mu_j \in \mathbb{R}^{784}$
- covariance matrix $\Sigma_j \in \mathbb{R}^{784 \times 784}$

Problem: formula for normal density uses Σ_j^{-1}, which is singular.

- Need to regularize: $\Sigma_j \to \Sigma_j + \sigma^2 I$
- This is a good idea even without the singularity issue

Error rate with regularization: **???**

Fisher's linear discriminant

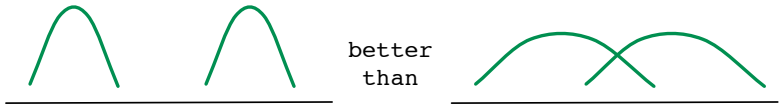
A framework for linear classification without Gaussian assumptions.

Use only first- and second-order statistics of the classes.

Class 1	Class 2
mean μ_1	mean μ_2
cov Σ_1	cov Σ_2
# pts n_1	# pts n_2

A linear classifier projects all data onto a direction w. Choose w so that:

- Projected means are well-separated, i.e. $(w \cdot \mu_1 - w \cdot \mu_2)^2$ is large.
- Projected within-class variance is small.



Fisher LDA (linear discriminant analysis)

Two classes: means μ_1, μ_2; covariances Σ_1, Σ_2; sample sizes n_1, n_2.

Project data onto direction (unit vector) w.

- Projected means: $w \cdot \mu_1$ and $w \cdot \mu_2$
- Projected variances: $w^T \Sigma_1 w$ and $w^T \Sigma_2 w$
- Average projected variance:

$$\frac{n_1(w^T \Sigma_1 w) + n_2(w^T \Sigma_2 w)}{n_1 + n_2} = w^T \Sigma w,$$

where $\Sigma = (n_1 \Sigma_1 + n_2 \Sigma_2)/(n_1 + n_2)$.

Find w to maximize $J(w) = \dfrac{(w \cdot \mu_1 - w \cdot \mu_2)^2}{w^T \Sigma w}$

Solution: $w \propto \Sigma^{-1}(\mu_1 - \mu_2)$. Look familiar?

Fisher LDA: proof

Goal: find w to maximize $J(w) = \dfrac{(w \cdot \mu_1 - w \cdot \mu_2)^2}{w^T \Sigma w}$

1. Assume Σ_1, Σ_2 are full rank; else project.
2. Since Σ_1 and Σ_2 are p.d., so is their weighted average, Σ.
3. Write $u = \Sigma^{1/2} w$. Then

$$\begin{aligned} \max_w \frac{(w^T(\mu_1 - \mu_2))^2}{w^T \Sigma w} &= \max_u \frac{(u^T \Sigma^{-1/2}(\mu_1 - \mu_2))^2}{u^T u} \\ &= \max_{u:\|u\|=1} (u \cdot (\Sigma^{-1/2}(\mu_1 - \mu_2)))^2 \end{aligned}$$

4. Solution: u is the unit vector in direction $\Sigma^{-1/2}(\mu_1 - \mu_2)$.
5. Therefore: $w = \Sigma^{-1/2} u \propto \Sigma^{-1}(\mu_1 - \mu_2)$.

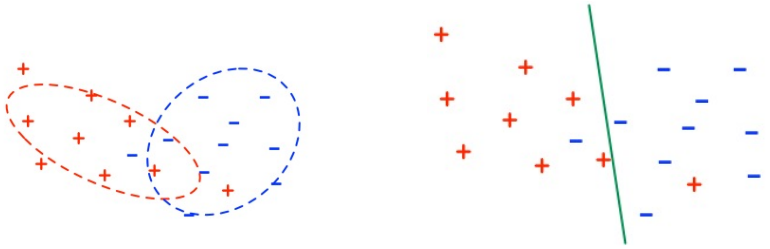
Classification with discriminative models

CSE 250B

Classification with parametrized models

Classifiers with a fixed number of parameters can represent a limited set of functions. Learning a model is about picking a good approximation.

Typically the x's are points in p-dimensional Euclidean space, $\mathbb{R}^p$.



Two ways to classify:

- **Generative**: model the individual classes.
- **Discriminative**: model the decision boundary between the classes.

Generative models: pros and cons

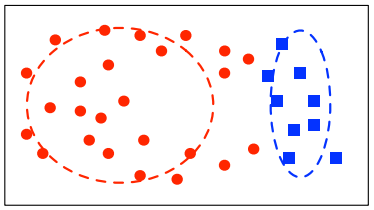
Advantages:

- Multiclass is a breeze
- Special density models (such as Bayes nets or hidden Markov models) can model temporal and other dependencies
- Returns not just a classification but also a confidence $\Pr(y|x)$
- For many common models: converges fast

Disadvantages:

- Formula for $\Pr(y|x)$ assumes the class-specific density models are perfect, but this is never true
- If we only care about classification, shouldn't we focus on the decision boundary rather than trying to model other aspects of the distribution of x?

Generative versus discriminative



The generative way:

- Fit: π_0, π_1, P_0, P_1
- This determines a full joint distribution $\Pr(x, y)$
- Use Bayes' rule to obtain $\Pr(y|x)$

Discriminative: model $\Pr(y|x)$ directly

The logistic regression model

What model to use for $\Pr(y|x)$?

- Say $\mathcal{Y} = \{-1, 1\}$. Recall: for Gaussians with common covariance,

$$\ln \frac{\Pr(y = 1 \mid x)}{\Pr(y = -1 \mid x)} = \underbrace{w \cdot x + \theta}_{\text{linear}}$$

- Can drop θ by adding an extra feature to x.
- Then $\Pr(y = 1 \mid x) = \Pr(y = -1 \mid x)\, e^{w \cdot x}$, whereupon

$$\Pr(y = -1 \mid x) = \frac{1}{1 + e^{w \cdot x}}$$

$$\Pr(y = 1 \mid x) = 1 - \frac{1}{1 + e^{w \cdot x}} = \frac{e^{w \cdot x}}{1 + e^{w \cdot x}} = \frac{1}{1 + e^{-w \cdot x}}$$

- More concisely,

$$\Pr(y \mid x) = \frac{1}{1 + e^{-y(w \cdot x)}}$$

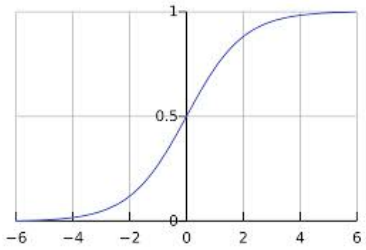
This is the **logistic regression model**, parametrized by w.

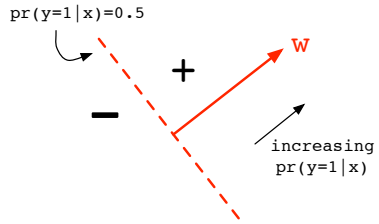
The squashing function

Take $\mathcal{X} = \mathbb{R}^p$ and $\mathcal{Y} = \{-1, 1\}$. The model specified by $w \in \mathbb{R}^p$ is

$$\Pr_w(y \mid x) = \frac{1}{1 + e^{-y(w \cdot x)}} = g(y(w \cdot x)),$$

where $g(z) = 1/(1 + e^{-z})$ is the *squashing function*.





Fitting w

The maximum-likelihood principle: given a data set

$$(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}) \in \mathbb{R}^p \times \{-1, 1\},$$

pick the $w \in \mathbb{R}^p$ that maximizes

$$\prod_{i=1}^{n} \Pr_w(y^{(i)} \mid x^{(i)}).$$

Easier to work with sums, so take log to get **loss function**

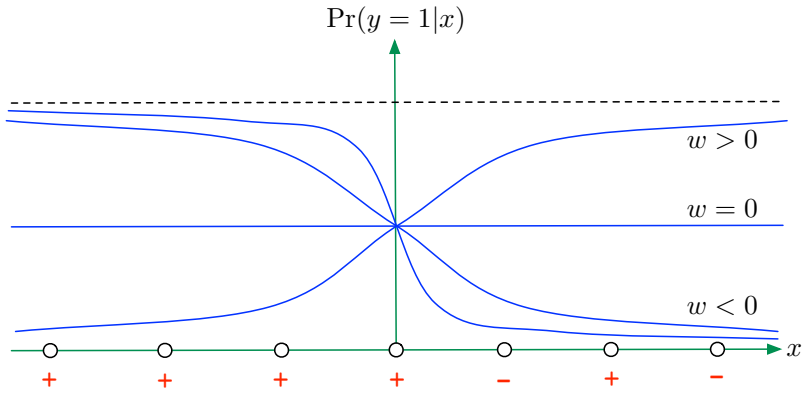
$$L(w) = -\sum_{i=1}^{n} \ln \Pr_w(y^{(i)} \mid x^{(i)}) = \sum_{i=1}^{n} \ln(1 + e^{-y^{(i)}(w \cdot x^{(i)})})$$

Our goal is to minimize $L(w)$.

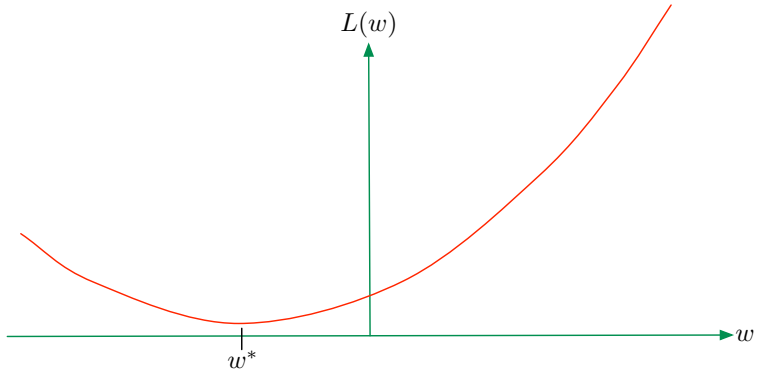
The good news: $L(w)$ is **convex** in w.

One dimensional example

$$\Pr_w(y \mid x) = \frac{1}{1 + e^{-ywx}}, \quad w \in \mathbb{R}$$



Example, cont'd



How to find the minimum of this convex function? A variety of options:

- Gradient descent
- Newton-Raphson

and many others.

Gradient descent procedure for logistic regression

Given $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}) \in \mathbb{R}^p \times \{-1, 1\}$, find

$$\arg\min_{w \in \mathbb{R}^p} L(w) = \sum_{i=1}^{n} \ln(1 + e^{-y^{(i)}(w \cdot x^{(i)})})$$

- Set $w_0 = 0$
- For $t = 0, 1, 2, \ldots$, until convergence:

$$w_{t+1} = w_t + \eta_t \sum_{i=1}^{n} y^{(i)} x^{(i)} \underbrace{\Pr_{w_t}(-y^{(i)}|x^{(i)})}_{\text{doubt}_t(x^{(i)}, y^{(i)})},$$

where η_t is a step size chosen by line search to minimize $L(w_{t+1})$.

Newton-Raphson procedure for logistic regression

- Set $w_0 = 0$
- For $t = 0, 1, 2, \ldots$, until convergence:

$$w_{t+1} = w_t + \eta_t (X^T D_t X)^{-1} \sum_{i=1}^{n} y^{(i)} x^{(i)} \Pr_{w_t}(-y^{(i)}|x^{(i)}),$$

where

- X is the $n \times p$ data matrix with one point per row
- D_t is an $n \times n$ diagonal matrix with (i, i) entry

$$D_{t,ii} = \Pr_{w_t}(1|x^{(i)}) \Pr_{w_t}(-1|x^{(i)})$$

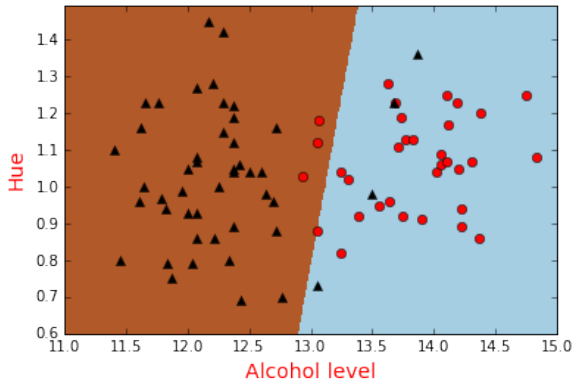
- η_t is a step size that is either fixed to 1 ("iterative reweighted least squares") or chosen by line search to minimize $L(w_{t+1})$.

Example: "wine" data set

Recall: data from three wineries from the same region of Italy.

- 13 attributes: hue, color intensity, flavanoids, ash content, ...
- 178 instances in all: split into 118 train, 60 test

Pick two classes and just two attributes (hue, alcohol content).

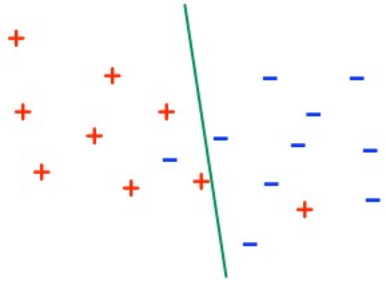


Test error using logistic regression: 10%.

More linear classification

CSE 250B

The decision boundary



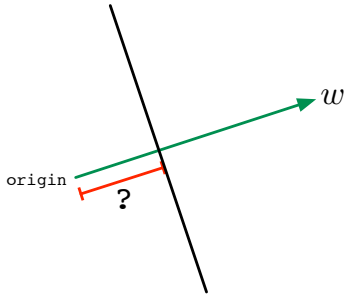
Decision boundary in $\mathbb{R}^p$ is a **hyperplane**.

- How is this boundary parametrized?
- How can we learn a hyperplane from training data?

Hyperplanes

Hyperplane $\{x : w \cdot x = b\}$

- orientation $w \in \mathbb{R}^p$
- offset $b \in \mathbb{R}$



Can always normalize w to unit length:

$$(w, b) \quad \longleftrightarrow \quad \left(\widehat{w} = \frac{w}{\|w\|}, \frac{b}{\|w\|}\right)$$

$$w \cdot x = b \quad \longleftrightarrow \quad \widehat{w} \cdot x = \frac{b}{\|w\|}$$

Equivalently: all points whose projection onto $\widehat{w}$ is $b/\|w\|$.

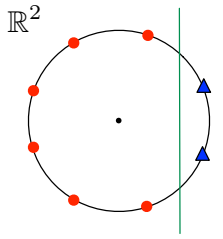
Homogeneous linear separators

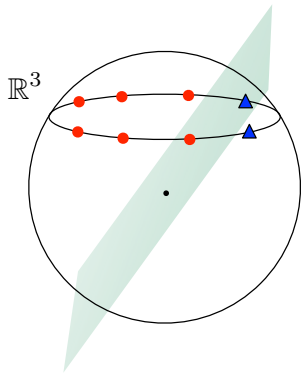
Hyperplanes that pass through the origin have no offset, $b = 0$.

Reduce to this case by adding an extra feature to x:

$$\widetilde{x} = (x, 1) \in \mathbb{R}^{p+1}$$

Then $\{x : w \cdot x = b\} \equiv \{x : \widetilde{w} \cdot \widetilde{x} = 0\}$ where $\widetilde{w} = (w, -b)$.





The learning problem: separable case

Input: training data $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}) \in \mathbb{R}^p \times \{-1, +1\}$

Output: linear classifier $w \in \mathbb{R}^p$ such that

$$y^{(i)}(w \cdot x^{(i)}) > 0 \quad \text{for } i = 1, 2, \ldots, n$$

This is linear programming:

- Each data point is a linear constraint on w
- Want to find w that satisfies all these constraints

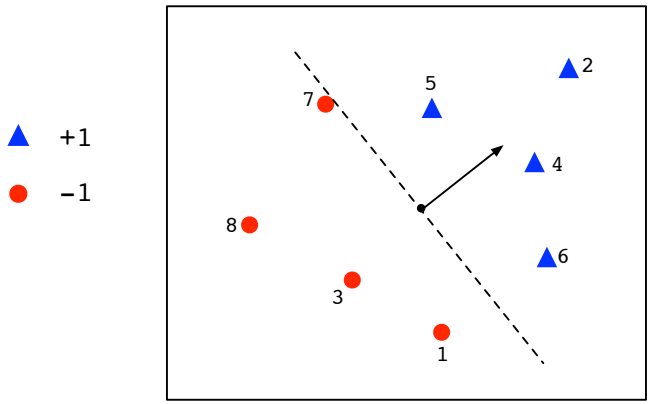
But we won't use generic linear programming methods, such as simplex.

A simple alternative: **Perceptron algorithm** (Rosenblatt, 1958)

- $w = 0$
- while some (x, y) is misclassified:
  - $w = w + yx$

Perceptron: example

- $w = 0$
- while some (x, y) is misclassified:
  - $w = w + yx$



Separator: $w = -x^{(1)} + x^{(6)}$

Perceptron: theory

Theorem: Let $R = \max \|x^{(i)}\|$. Suppose there is a unit vector w^* and some (margin) $\gamma > 0$ such that

$$y^{(i)}(w^* \cdot x^{(i)}) \geq \gamma \quad \text{for all } i.$$

Then the Perceptron algorithm converges after at most R^2/γ^2 updates.

Proof idea: track $\cos(\angle(w, w^*)) = (w \cdot w^*)/\|w\|$.

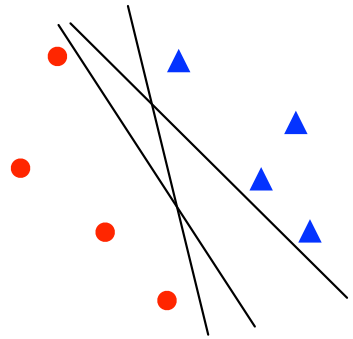
1. Let w_t be the classifier vector after t updates. So $w_0 = 0$.
2. If the $(t+1)$st misclassified point is (x, y), then $w_{t+1} = w_t + yx$ and

$$w_{t+1} \cdot w^* = (w_t + yx) \cdot w^* = w_t \cdot w^* + y(w^* \cdot x) \geq w_t \cdot w^* + \gamma$$
$$\|w_{t+1}\|^2 = \|w_t + yx\|^2 = \|w_t\|^2 + \|x\|^2 + 2 \underbrace{y(w_t \cdot x)}_{<0 \text{ (misclass.)}} \leq \|w_t\|^2 + R^2$$

3. $\therefore$ After T updates, we have $w_T \cdot w^* \geq T\gamma$ and $\|w_T\|^2 \leq TR^2$.
4. Cauchy-Schwarz: $w_T \cdot w^* \leq \|w_T\|$. So $T\gamma \leq \sqrt{TR^2} \Rightarrow T \leq R^2/\gamma^2$.

A better separator?

For a linearly separable data set, there are in general many possible separating hyperplanes, and Perceptron is guaranteed to find one of them.



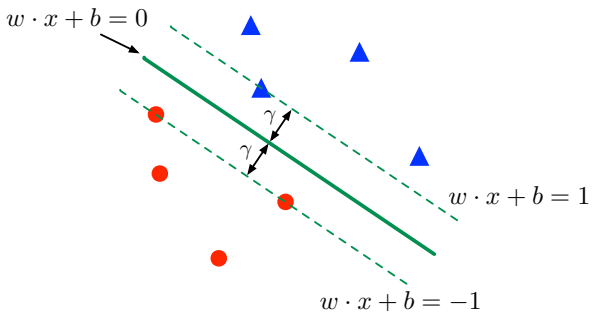
But is there a better, more systematic choice of separator? The one with the most buffer around it, for instance?

Maximizing the margin

Given training data $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}) \in \mathbb{R}^p \times \{-1, +1\}$, find $w \in \mathbb{R}^p$ and $b \in \mathbb{R}$ such that $y^{(i)}(w \cdot x^{(i)} + b) > 0$ for all i.

By scaling w, b, can equally ask for

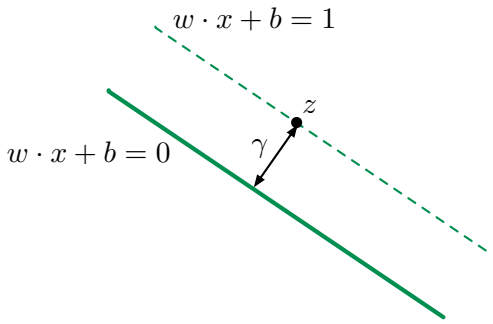
$$y^{(i)}(w \cdot x^{(i)} + b) \geq 1 \quad \text{for all } i.$$



Maximize the **margin** γ.

What is the margin?

Close-up of a point z on the positive boundary.



Let $\widehat{w}$ be the unit vector in the direction of w, i.e. $\widehat{w} = w/\|w\|$.
Then $z - \gamma\widehat{w}$ is on the separator, so

$$w \cdot (z - \gamma\widehat{w}) + b = 0 \quad \Rightarrow \quad \gamma w \cdot \widehat{w} = w \cdot z + b = 1 \quad \Rightarrow \quad \gamma = 1/\|w\|$$

In short: to maximize the margin, minimize $\|w\|$.

Maximum-margin linear classifier

- Given $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}) \in \mathbb{R}^p \times \{-1, +1\}$.

$$\begin{array}{ll} \text{(PRIMAL)} & \min\limits_{w \in \mathbb{R}^p, b \in \mathbb{R}} \ \dfrac{1}{2}\|w\|^2 \\ \text{s.t.:} & y^{(i)}(w \cdot x^{(i)} + b) \geq 1 \quad \text{for all } i = 1, 2, \ldots, n \end{array}$$

- This is a convex optimization problem:
  - Convex objective function
  - Linear constraints
- It has a dual maximization problem with the same optimum value.

$$\begin{array}{ll} \text{(DUAL)} & \max\limits_{\alpha \in \mathbb{R}^n} \ \displaystyle\sum_{i=1}^n \alpha_i - \sum_{i,j=1}^n \alpha_i \alpha_j y^{(i)} y^{(j)} (x^{(i)} \cdot x^{(j)}) \\ & \text{s.t.:} \ \displaystyle\sum_{i=1}^n \alpha_i y^{(i)} = 0 \\ & \alpha \geq 0 \end{array}$$

Complementary slackness

$$\begin{array}{ll} \text{(PRIMAL)} & \min\limits_{w \in \mathbb{R}^p, b \in \mathbb{R}} \ \dfrac{1}{2}\|w\|^2 \\ \text{s.t.:} & y^{(i)}(w \cdot x^{(i)} + b) \geq 1 \quad \text{for all } i = 1, 2, \ldots, n \end{array}$$

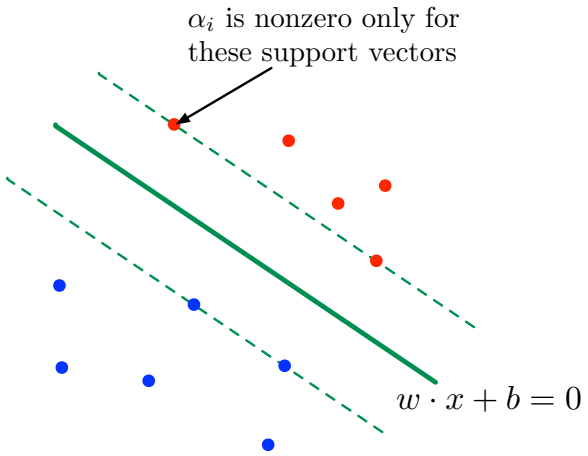
$$\begin{array}{ll} \text{(DUAL)} & \max\limits_{\alpha \in \mathbb{R}^n} \ \displaystyle\sum_{i=1}^n \alpha_i - \sum_{i,j=1}^n \alpha_i \alpha_j y^{(i)} y^{(j)} (x^{(i)} \cdot x^{(j)}) \\ & \text{s.t.:} \ \displaystyle\sum_{i=1}^n \alpha_i y^{(i)} = 0 \\ & \alpha \geq 0 \end{array}$$

At optimality, $w = \sum_{i=1}^n \alpha_i y^{(i)} x^{(i)}$ and moreover

$$\alpha_i > 0 \quad \Rightarrow \quad y^{(i)}(w \cdot x^{(i)} + b) = 1$$

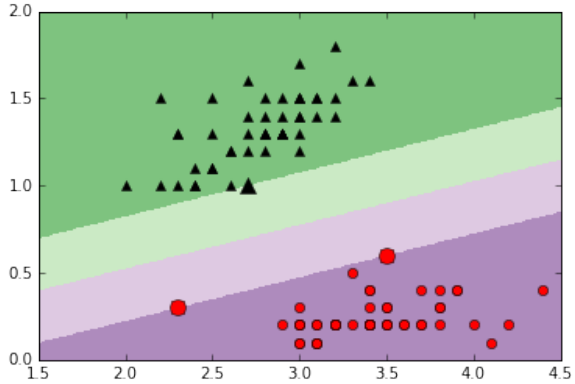
Points $x^{(i)}$ with $\alpha_i > 0$ are called **support vectors**.

Support vectors



Linear classifier $w = \sum_{i=1}^{n} \alpha_i y^{(i)} x^{(i)}$ is a function of just the support vectors.

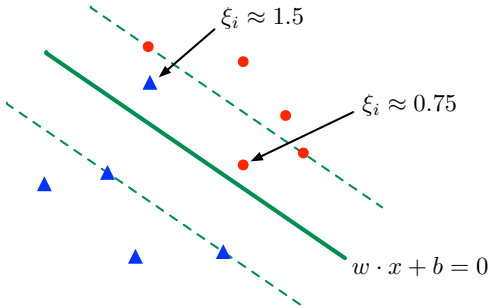
Small example: Iris data set



The non-separable case

Idea: allow each data point $x^{(i)}$ some slack ξ_i.

$$\begin{aligned} &\text{(PRIMAL)} && \min_{w \in \mathbb{R}^p, b \in \mathbb{R}, \xi \in \mathbb{R}^n} \ \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i \\ &\text{s.t.:} && y^{(i)}(w \cdot x^{(i)} + b) \geq 1 - \xi_i \quad \text{for all } i = 1, 2, \ldots, n \\ & && \xi \geq 0 \end{aligned}$$



Dual for general case

$$\begin{aligned} &\text{(PRIMAL)} && \min_{w \in \mathbb{R}^p, b \in \mathbb{R}, \xi \in \mathbb{R}^n} \ \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i \\ &\text{s.t.:} && y^{(i)}(w \cdot x^{(i)} + b) \geq 1 - \xi_i \quad \text{for all } i = 1, 2, \ldots, n \\ & && \xi \geq 0 \end{aligned}$$

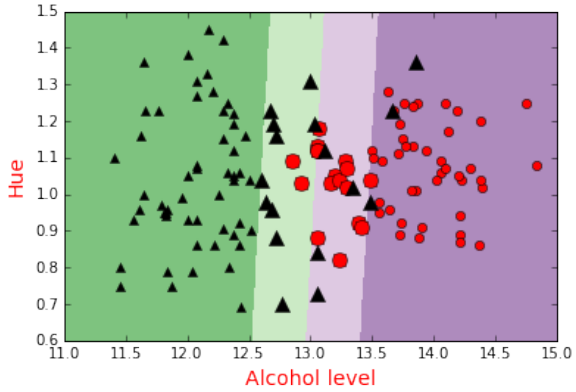
$$\begin{aligned} &\text{(DUAL)} && \max_{\alpha \in \mathbb{R}^n} \ \sum_{i=1}^{n} \alpha_i - \sum_{i,j=1}^{n} \alpha_i \alpha_j y^{(i)} y^{(j)} (x^{(i)} \cdot x^{(j)}) \\ &\text{s.t.:} && \sum_{i=1}^{n} \alpha_i y^{(i)} = 0 \\ & && 0 \leq \alpha_i \leq C \end{aligned}$$

At optimality, $w = \sum_i \alpha_i y^{(i)} x^{(i)}$, with

$$\begin{aligned} 0 < \alpha_i < C \quad &\Rightarrow \quad y^{(i)}(w \cdot x^{(i)} + b) = 1 \\ \alpha_i = C \quad &\Rightarrow \quad y^{(i)}(w \cdot x^{(i)} + b) = 1 - \xi_i \end{aligned}$$

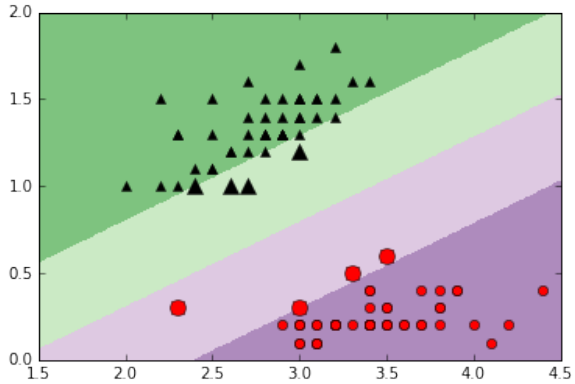
Wine data set

Here $C = 1.0$



Back to Iris

$C = 1$



Convex surrogates for 0-1 loss

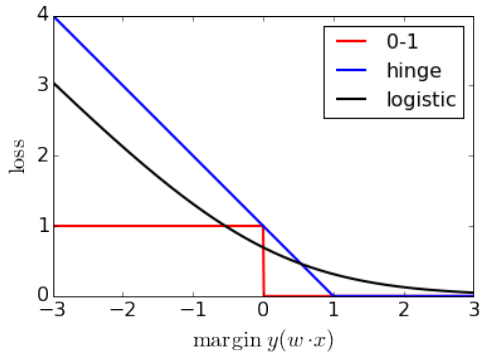
Want a separator w that misclassifies as few training points as possible.

- 0-1 loss: charge $1(y(w \cdot x) < 0)$ for each (x, y)

Problem: this is NP-hard.

Instead, use **convex** loss functions.

- Hinge loss (SVM): charge $(1 - y(w \cdot x))_+$
- Logistic loss: charge $\ln(1 + e^{-y(w \cdot x)})$



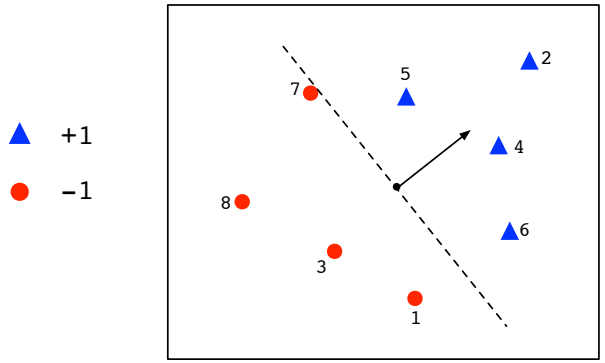
Kernels

CSE 250B

Recall: Perceptron

Input space $\mathcal{X} = \mathbb{R}^p$, label space $\mathcal{Y} = \{-1, 1\}$

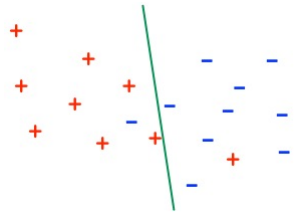
- $w = 0$
- while some (x, y) is misclassified:
  - $w = w + yx$



Separator: $w = -x^{(1)} + x^{(6)}$

Deviations from linear separability

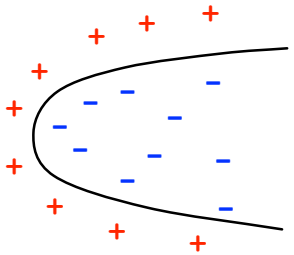
Noise



Find a separator that minimizes a convex loss function related to the number of mistakes.

e.g. SVM, logistic regression.

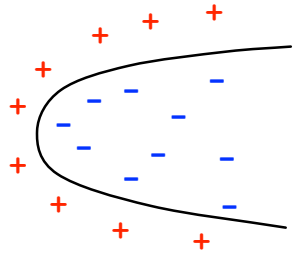
Systematic deviation



What to do with this?

Systematic inseparability

In this case, the actual boundary looks quadratic.



Quick fix: in addition to the regular features $x = (x_1, x_2, \ldots, x_p)$, add in extra features:

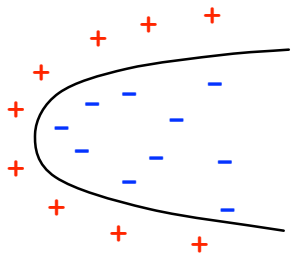
$$x_1^2,\ x_2^2,\ \ldots,\ x_p^2$$

$$x_1x_2,\ x_1x_3,\ \ldots,\ x_{p-1}x_p$$

The new, enhanced data vectors are of the form:

$$\Phi(x) = (1, \sqrt{2}x_1, \ldots, \sqrt{2}x_p, x_1^2, \ldots, x_p^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_{p-1}x_p).$$

Adding new features



Actual boundary is something like $x_1 = x_2^2 + 5$.

- This is quadratic in $x = (1, x_1, x_2)$
- But it is linear in $\Phi(x) = (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2)$

By embedding the data in a higher-dimensional feature space, we can keep using a linear classifier!

Perceptron revisited

Learning in the higher-dimensional feature space:

- $w = 0$
- while some $y(w \cdot \Phi(x)) < 0$:
  - $w = w + y\,\Phi(x)$

Final w is a weighted linear sum of various $\Phi(x)$.

Problem: number of features has now increased dramatically.
For MNIST, from 784 to 307720!

The **kernel trick** (Aizenman, Braverman, Rozonoer (1964)):

- The only time we ever access $\Phi(x)$ is to compute $w \cdot \Phi(x)$. If, say,

$$w = a_1\Phi(x^{(1)}) + a_2\Phi(x^{(2)}) + a_3\Phi(x^{(3)}),$$

  then $w \cdot \Phi(x)$ is a weighted sum of dot products $\Phi(x) \cdot \Phi(x^{(i)})$.
- Can we compute such dot products without writing out the $\Phi(x)$'s?

Computing dot products

In 2-d:

$$\begin{aligned}
&\Phi(x) \cdot \Phi(z) \\
&= (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2) \cdot (1, \sqrt{2}z_1, \sqrt{2}z_2, z_1^2, z_2^2, \sqrt{2}z_1z_2) \\
&= 1 + 2x_1z_1 + 2x_2z_2 + x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 \\
&= (1 + x_1z_1 + x_2z_2)^2 \quad = \quad (1 + x \cdot z)^2
\end{aligned}$$

In p dimensions:

$$\begin{aligned}
&\Phi(x) \cdot \Phi(z) \\
&= (1, \sqrt{2}x_1, \ldots, \sqrt{2}x_p, x_1^2, \ldots, x_p^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_{p-1}x_p) \cdot \\
&\quad (1, \sqrt{2}z_1, \ldots, \sqrt{2}z_p, z_1^2, \ldots, z_p^2, \sqrt{2}z_1z_2, \ldots, \sqrt{2}z_{p-1}z_p) \\
&= 1 + 2\sum_i x_iz_i + \sum_i x_i^2z_i^2 + 2\sum_{i \neq j} x_ix_jz_iz_j \\
&= (1 + x_1z_1 + \cdots + x_pz_p)^2 \quad = \quad (1 + x \cdot z)^2
\end{aligned}$$

For MNIST: computing dot products in the 307720-dimensional quadratic feature space takes time proportional to just 784, the original dimension!

The kernel trick

Why does it work?

1. The only time we ever look at data – during training or subsequent classification – is to compute dot products $w \cdot \Phi(x)$.
2. And w itself is a linear combination of $\Phi(x)$'s. If, say,

$$w = \alpha_1\Phi(x^{(1)}) + \alpha_{22}\Phi(x^{(22)}) + \alpha_{37}\Phi(x^{(37)}),$$

   we can store w as $[(1, \alpha_1), (22, \alpha_{22}), (37, \alpha_{37})]$.
3. Dot products $\Phi(x) \cdot \Phi(x')$ can be computed efficiently, without ever writing out the high-dimensional embedding $\Phi(\cdot)$.

Kernel Perceptron

Learning from data $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}) \in \mathcal{X} \times \{-1, 1\}$

Primal form:

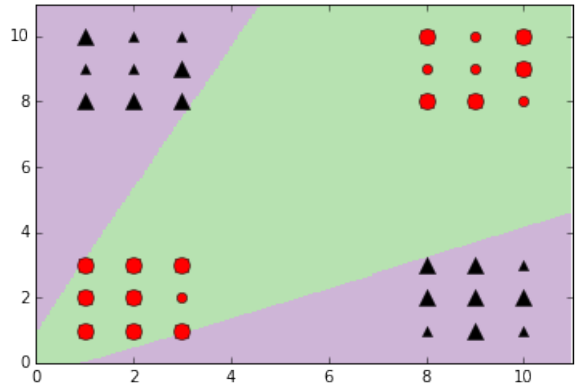
- $w = 0$
- while there is some i with $y^{(i)}(w \cdot \Phi(x^{(i)})) < 0$:
  - $w = w + y^{(i)}\,\Phi(x^{(i)})$

Dual form: $w = \sum_j \alpha_j y^{(j)} \Phi(x^{(j)})$, where $\alpha \in \mathbb{R}^n$.

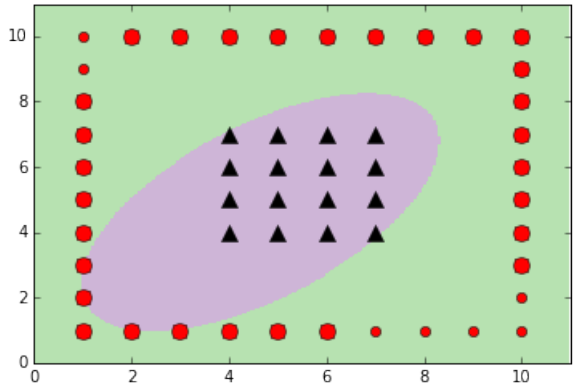
- $\alpha = 0$
- while there is some i with $y^{(i)}\left(\sum_j \alpha_j y^{(j)} \underbrace{\Phi(x^{(j)}) \cdot \Phi(x^{(i)})}_{\text{efficient}}\right) < 0$:
  - $\alpha_i = \alpha_i + 1$

To classify a new point x: Return sign $\left(\sum_j \alpha_j y^{(j)} \Phi(x^{(j)}) \cdot \Phi(x)\right)$.

Kernel Perceptron: Examples



Kernel Perceptron: Examples



Does this work with SVMs?

$$\text{(PRIMAL)} \quad \min_{w \in \mathbb{R}^p, b \in \mathbb{R}, \xi \in \mathbb{R}^n} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^n \xi_i$$

$$\text{s.t.: } y^{(i)}(w \cdot x^{(i)} + b) \geq 1 - \xi_i \quad \text{for all } i = 1, 2, \ldots, n$$

$$\xi \geq 0$$

$$\text{(DUAL)} \quad \max_{\alpha \in \mathbb{R}^n} \sum_{i=1}^n \alpha_i - \sum_{i,j=1}^n \alpha_i \alpha_j y^{(i)} y^{(j)} (x^{(i)} \cdot x^{(j)})$$

$$\text{s.t.: } \sum_{i=1}^n \alpha_i y^{(i)} = 0$$

$$0 \leq \alpha_i \leq C$$

At optimality, $w = \sum_i \alpha_i y^{(i)} x^{(i)}$, with

$$0 < \alpha_i < C \quad \Rightarrow \quad y^{(i)}(w \cdot x^{(i)} + b) = 1$$

$$\alpha_i = C \quad \Rightarrow \quad y^{(i)}(w \cdot x^{(i)} + b) = 1 - \xi_i$$

Kernel SVM

1. **Embedding.** Pick a mapping $x \mapsto \Phi(x)$.
2. **Learning.** Solve the dual problem:

$$\max_{\alpha \in \mathbb{R}^n} \sum_{i=1}^{n} \alpha_i - \sum_{i,j=1}^{n} \alpha_i \alpha_j y^{(i)} y^{(j)} \underbrace{(\Phi(x^{(i)}) \cdot \Phi(x^{(j)}))}_{\text{efficient}}$$

$$\text{s.t.:} \quad \sum_{i=1}^{n} \alpha_i y^{(i)} = 0$$

$$0 \leq \alpha_i \leq C$$

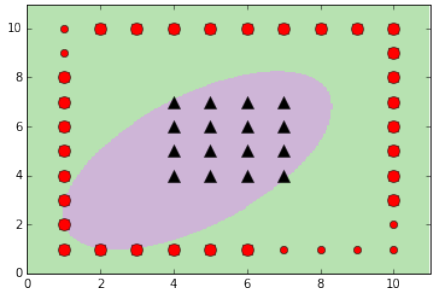
   This yields $w = \sum_i \alpha_i y^{(i)} \Phi(x^{(i)})$.
   Offset b is obtained from the complementary slackness conditions.

3. **Classification.** Given a new point x, classify as

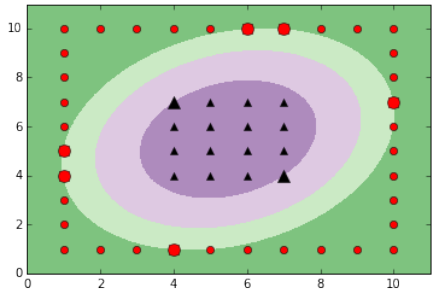
$$\text{sign}\left(\sum_i \alpha_i y^{(i)} \underbrace{(\Phi(x^{(i)}) \cdot \Phi(x))}_{\text{again}} + b \right).$$

Kernel Perceptron vs. Kernel SVM: examples

Perceptron:

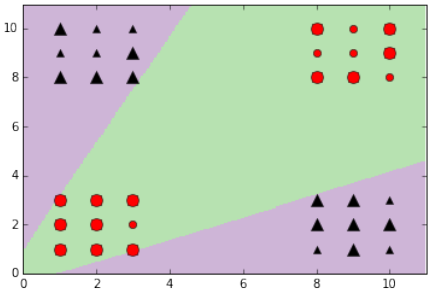


SVM:

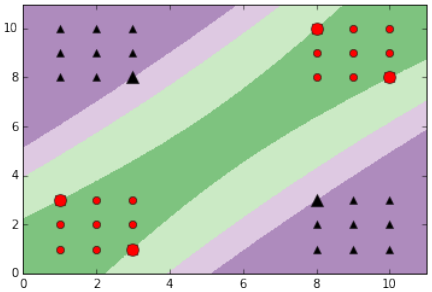


Kernel Perceptron vs. Kernel SVM: examples

Perceptron:

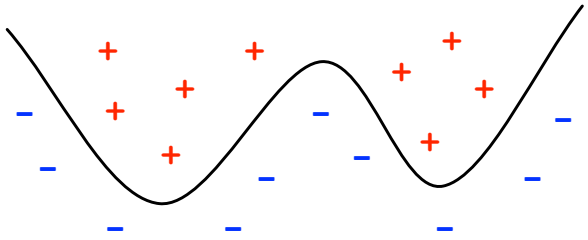


SVM:



Polynomial decision boundaries

To get a decision surface which is an arbitrary polynomial of order d:



- Let $\Phi(x)$ consist of all terms of order $\leq d$, such as $x_1 x_2^2 x_3^{d-3}$.
  (How many such terms are there, roughly?)
- Same trick works: $\Phi(x) \cdot \Phi(z) = (1 + x \cdot z)^d$.

The **kernel function**, a measure of similarity:

$$k(x, z) = \Phi(x) \cdot \Phi(z).$$

String kernels

Sequence data:

- text documents
- speech signals
- protein sequences

Each data point is a sequence of arbitrary length. This yields input spaces like:

$$\mathcal{X} = \{A, C, G, T\}^*$$
$$\mathcal{X} = \{\text{English words}\}^*$$

What kind of embedding $\Phi(x)$ is suitable for variable-length sequences x?

We will use an infinite-dimensional embedding!

String kernels, cont'd

For each substring s, define *feature*:

$$\Phi_s(x) = \#\text{ of times substring } s \text{ appears in } x$$

and let $\Phi(x)$ be a vector with one coordinate for each possible string:

$$\Phi(x) = (\Phi_s(x) : \text{all strings } s).$$

Example: the embedding of "aardvark" includes features

$$\Phi_{\text{ar}}(\text{aardvark}) = 2,\ \Phi_{\text{th}}(\text{aardvark}) = 0, \ldots$$

A linear classifier based on such features is potentially very powerful.

We can compute dot products fast! To compute $k(x, z) = \Phi(x) \cdot \Phi(z)$:

for each substring s of x: count how often s appears in z

Using dynamic programming, this takes time $O(|x| \cdot |z|)$.

The kernel function

Now shift attention:

- away from the embedding $\Phi(x)$, which we never explicitly construct,
- towards the thing we actually use: the **similarity measure** $k(x, z)$

Rewrite learning algorithm and final classifier in terms of k.

The classifier $w \cdot \Phi(x)$, for

$$w = \alpha_1 y^{(1)} \Phi(x^{(1)}) + \cdots + \alpha_n y^{(n)} \Phi(x^{(n)}),$$

becomes a **similarity-weighted vote**,

$$F(x) = \alpha_1 y^{(1)} k(x^{(1)}, x) + \cdots + \alpha_n y^{(n)} k(x^{(n)}, x).$$

Can we choose k to be any similarity function suitable for the application domain?

Mercer's condition

A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a valid kernel function if it corresponds to some embedding: that is, if there exists Φ defined on $\mathcal{X}$ such that

$$k(x, z) = \Phi(x) \cdot \Phi(z).$$

Mercer (1909): This is equivalent to requiring that for any finite subset $\{x^{(1)}, \ldots, x^{(m)}\} \subset \mathcal{X}$, the $m \times m$ similarity matrix

$$K_{ij} = k(x^{(i)}, x^{(j)})$$

is positive semidefinite.

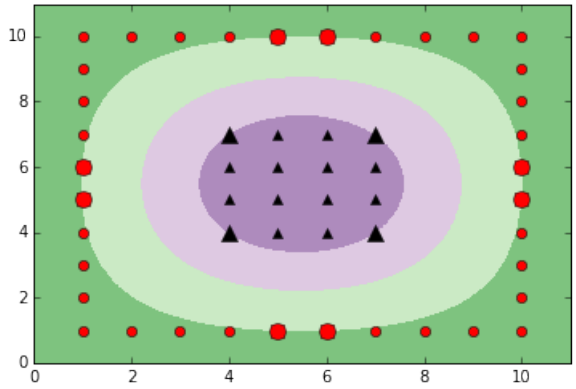
Do you see why this matrix is p.s.d. if $k(x, z) = \Phi(x) \cdot \Phi(z)$?

A popular similarity function: the **Gaussian kernel** or **RBF kernel**

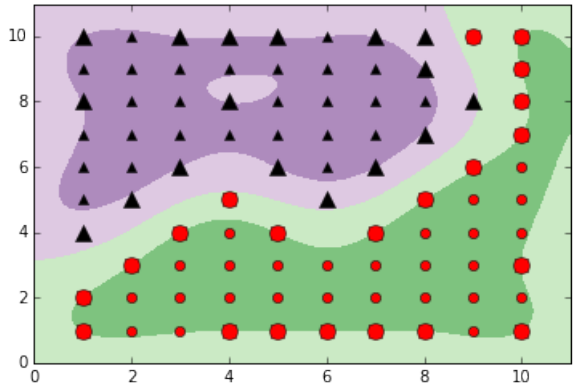
$$k(x, z) = e^{-\|x-z\|^2/2\sigma^2},$$

where σ is an adjustable scale parameter.

RBF kernel: examples



RBF kernel: examples



Kernels: postscript

1. Customized kernels
   - For many different domains (NLP, biology, speech, ...)
   - Over many different structures (sequences, sets, trees, graphs, ...)
2. Learning the kernel function
   Given a set of plausible kernels, find a linear combination of them that works well.
3. Speeding up learning and prediction
   The $n \times n$ kernel matrix $k(x^{(i)}, x^{(j)})$ is a bottleneck for large n.
   One idea:
   - Go back to the primal space!
   - Replace the embedding Φ by a low-dimensional mapping $\widetilde{\Phi}$ such that

$$\widetilde{\Phi}(x) \cdot \widetilde{\Phi}(z) \approx \Phi(x) \cdot \Phi(z).$$

   This can be done, for instance, by writing Φ in the Fourier basis and then randomly sampling features.

Richer output spaces

CSE 250B

Multiclass classification

We have mostly discussed binary classification problems, with $|\mathcal{Y}| = 2$. Do the methods we've studied generalize to cases with $k > 2$ labels?

- Nearest neighbor?
- Decision trees?
- Generative models?
- Linear classifiers?

Linear classifiers seem inherently binary: there are just two sides of the boundary! How can they be extended to multiple classes?

Multiclass logistic regression

Binary logistic regression: for $\mathcal{X} = \mathbb{R}^p$, the classifier is given by $w \in \mathbb{R}^p$:

$$\Pr(y = 1|x) = \frac{e^{w \cdot x}}{1 + e^{w \cdot x}}.$$

When $\mathcal{Y} = \{1, 2, \ldots, k\}$, specify a classifier by $w_1, \ldots, w_k \in \mathbb{R}^p$:

$$\Pr(y = j|x) = \frac{e^{w_j \cdot x}}{e^{w_1 \cdot x} + \cdots + e^{w_k \cdot x}}.$$

Prediction: given a point x, predict label

$$\arg\max_j \Pr(y = j|x) \;=\; \arg\max_j \; w_j \cdot x$$

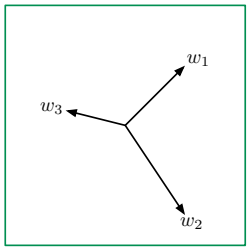
Learning: given data $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}) \in \mathbb{R}^p \times \mathcal{Y}$, find vectors $w_1, \ldots, w_k \in \mathbb{R}^p$ that maximize the likelihood

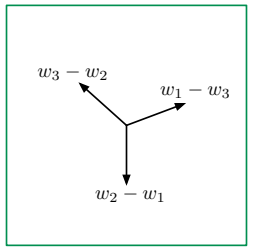
$$\prod_{i=1}^{n} \Pr(y^{(i)}|x^{(i)}).$$

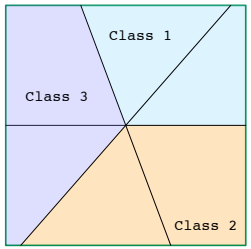
Taking negative log gives a convex minimization problem.

Multiclass prediction with linear functions

- $\mathcal{X} = \mathbb{R}^p$ and $\mathcal{Y} = \{1, 2, \ldots, k\}$.
- **Model:** $w_1, \ldots, w_k \in \mathbb{R}^p$, one per class.
- **Prediction:** On instance x, predict label $\arg\max_j w_j \cdot x$.







Each class is the intersection of half-spaces through the origin.

Multiclass Perceptron

Setting: $\mathcal{X} = \mathbb{R}^p$ and $\mathcal{Y} = \{1, 2, \ldots, k\}$

Model: $w_1, \ldots, w_k \in \mathbb{R}^p$, one per class.

Prediction: On instance x, predict label $\arg\max_j w_j \cdot x$.

Learning. Given training set $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$:

- Initialize $w_1 = \cdots = w_k = 0$
- Repeat while some training point (x, y) is misclassified:

$$\begin{aligned} \text{for correct label } y: &\quad w_y = w_y + x \\ \text{for predicted label } \widehat{y}: &\quad w_{\widehat{y}} = w_{\widehat{y}} - x \end{aligned}$$

Guarantee: Suppose all $\|x^{(i)}\| \leq R$ and that there exist unit-length $u_1, \ldots, u_k \in \mathbb{R}^p$ and "margin" $\gamma > 0$ such that for all i and all $y \neq y^{(i)}$,

$$u_{y^{(i)}} \cdot x^{(i)} - u_y \cdot x^{(i)} \;\geq\; \gamma.$$

Then the multiclass perceptron algorithm makes at most $2kR^2/\gamma^2$ updates.

Multiclass SVM

Model: $w_1, \ldots, w_k \in \mathbb{R}^p$, one per class.

Prediction: On instance x, predict label $\arg\max_j w_j \cdot x$.

Learning. Given $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}) \in \mathbb{R}^p \times \{1, \ldots, k\}$:

$$\min_{w_1,\ldots,w_k \in \mathbb{R}^p, \xi \in \mathbb{R}^n} \quad \frac{1}{2}\sum_{j=1}^{k} \|w_j\|^2 + C\sum_{i=1}^{n} \xi_i$$

$$\text{s.t.:} \quad w_{y^{(i)}} \cdot x^{(i)} - w_y \cdot x^{(i)} \geq 1 - \xi_i \quad \text{for all } i \text{ and all } y \neq y^{(i)}$$

$$\xi \geq 0$$

Once again, a convex optimization problem.

Structured output spaces: examples

Part-of-speech tagging.

the/D cat/N bit/V the/D dog/N

To score a candidate tagging y of a sentence x, add up:

- Score for each (word, tag)
- Score for each trigram (tag1, tag2, tag3)
- Other such component scores

Inaccurate to treat each tag as a separate prediction problem.

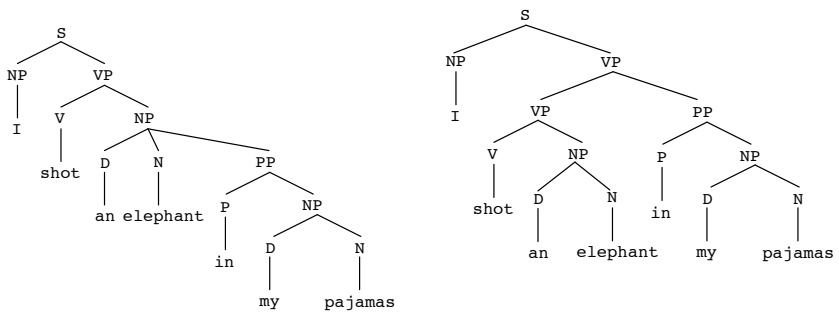
To tag a given sentence x: find the tagging y with maximum score. Can be done efficiently by dynamic programming.

Structured output spaces: examples

Parsing.

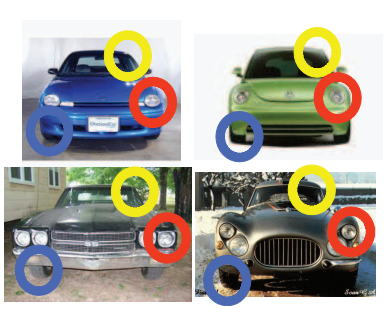
Groucho Marx (1930): While hunting in Africa, I shot an elephant in my pajamas. How an elephant got into my pajamas I'll never know.

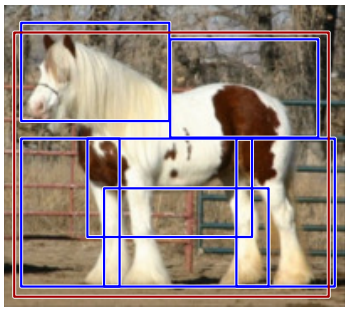
Here are two possible parse trees y for the sentence $x =$ "I shot an elephant in my pajamas".



Structured output spaces: examples

Parts-based object recognition.





Structured output prediction

How to handle such output spaces $\mathcal{Y}$?

- Features based on both the input and output.
  For any instance (e.g. sentence) x and candidate output (e.g. POS tagging) y, let

  $$\phi_1(x,y), \phi_2(x,y), \ldots, \phi_k(x,y)$$

  be features that give a sense of whether y is a desirable output for x. For instance: all word-tag pairs and tag trigrams.
  Package these features into a vector:

  $$\Phi(x,y) = (\phi_1(x,y), \phi_2(x,y), \ldots, \phi_k(x,y))$$

- Score outputs based on a linear function of the features.
  The score for output $y \in \mathcal{Y}$ is $w \cdot \Phi(x,y)$, where $w \in \mathbb{R}^k$.
- Predict the highest-scoring output.
  For instance x, return $\arg\max_y w \cdot \Phi(x,y)$. This can often be done efficiently with dynamic programming.

Learning task: given data, find a suitable weight vector w.

Structured-output Perceptron

Given training data $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}) \in \mathcal{X} \times \mathcal{Y}$:

- Initialize $w = 0$
- Repeat until satisfied:
  - For $i = 1$ to n:

$$\text{Prediction:} \quad \widehat{y} = \arg\max_y w \cdot \Phi(x^{(i)}, y)$$

$$\text{If } y^{(i)} \neq \widehat{y}: \quad w = w + \Phi(x^{(i)}, y^{(i)}) - \Phi(x^{(i)}, \widehat{y})$$

Convergence guarantee under a margin condition, as before.

Structured-output SVM

Loss function.
Not all errors are equal, especially when the outputs have many parts.
Let $\Delta(y, \widehat{y})$ be the loss when predicting $\widehat{y}$ instead of y.

Learning. Given $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}) \in \mathcal{X} \times \mathcal{Y}$:

$$\min_{w \in \mathbb{R}^k, \xi \in \mathbb{R}^n} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i$$

$$w \cdot \Phi(x^{(i)}, y^{(i)}) - w \cdot \Phi(x^{(i)}, y) \geq \Delta(y^{(i)}, y) - \xi_i \quad \text{for all } i \text{ and all } y \neq y^{(i)}$$

$$\xi \geq 0$$

Clever optimization tricks are needed to solve this efficiently.

Boosting

CSE 250B

Weak learners

It is often easy to come up with a **weak classifier**, one that is only slightly better than random guessing.

$$\Pr(h(X) \neq Y) = \frac{1}{2} - \epsilon$$

A learning algorithm that can consistently generate such classifiers is called a **weak learner**.

Is it possible to systematically boost the quality of a weak learner?

Blueprint:

- Think of the data set $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$ as a distribution D on $\mathcal{X} \times \mathcal{Y}$
- Repeat for $t = 1, 2, \ldots$:
  - Feed D to the weak learner, get back a weak classifier h_t
  - Reweight D to put more emphasis on points that h_t gets wrong
- Combine all these h_t's somehow

Adaboost

Assume $\mathcal{Y} = \{-1, 1\}$. Given: $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}) \in \mathcal{X} \times \mathcal{Y}$.

1. Initialize $D_1(i) = 1/n$ for all $i = 1, 2, \ldots, n$
2. For $t = 1, 2, \ldots, T$:
   - Give D_t to weak learner, get back some $h_t : \mathcal{X} \to [-1, 1]$
   - Update distribution:

$$r_t = \sum_{i=1}^{n} D_t(i) y^{(i)} h_t(x^{(i)}) \in [-1, 1] \quad (h_t\text{'s margin of success})$$

$$\alpha_t = \frac{1}{2} \ln \frac{1 + r_t}{1 - r_t} \in \mathbb{R} \quad \text{(strength of update)}$$

$$D_{t+1}(i) \propto D_t(i) \exp\left(-\alpha_t y^{(i)} h_t(x^{(i)})\right)$$

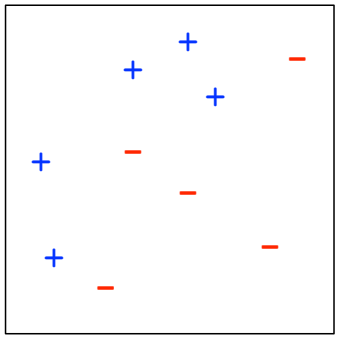
3. Final classifier:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

Example

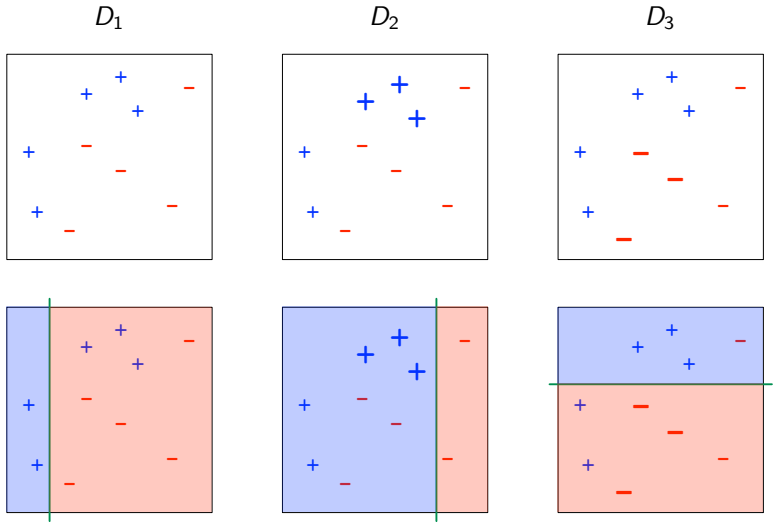
(From Freund and Schapire's tutorial.)

Training set:

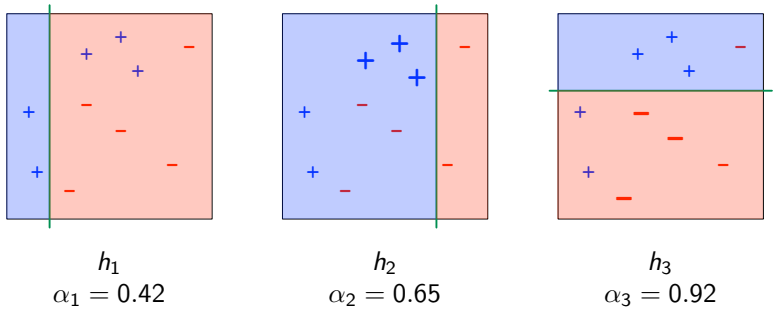


Weak classifiers: single-coordinate thresholds, popularly known as "decision stumps" (in this case, horizontal and vertical lines)

Example, cont'd

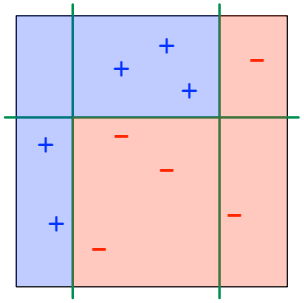


Example, cont'd



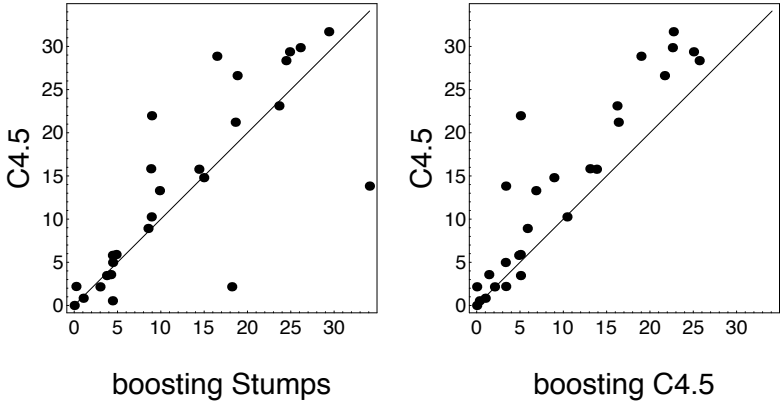
Final classifier:

$\text{sign}(0.42h_1(x)+0.65h_2(x)+0.92h_3(x))$



Boosting versus decision tree

Freund and Schapire: results on 27 benchmark data sets from the UCI repository.



Training error dropoff

The surprising power of a weak learner:

Theorem. Suppose that on each round t, the weak learner returns a classifier h_t that differs from random by some margin γ:

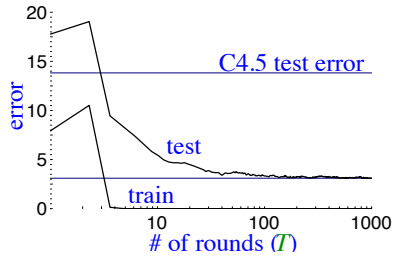
$$\left| \sum_{i=1}^{n} D_t(i) y^{(i)} h_t(x^{(i)}) \right| \geq \gamma.$$

Then after T rounds the training error is at most $e^{-\gamma^2 T/2}$.

Presumably, there will come a time T at which further reductions in training error are simply overfitting and will cause test error to rise?

Overfitting?

Freund and Schapire: boosting decision trees for "letter" dataset.



- Test error does not increase, even after 1000 rounds (total size over 2 million nodes)
- Test error keeps dropping even after training error has gone to zero:

	# rounds		
	5	100	1000
train error	0.0	0.0	0.0
test error	8.4	3.3	3.1

Looking at the margin

Recall the final classifier (with weights normalized to sum to 1):

$$H(x) = \text{sign}\underbrace{\left(\frac{\sum_t \alpha_t h_t(x)}{\sum_t |\alpha_t|}\right)}_{\text{call this } f(x)}$$

The **margin** of this classifier on a particular $(x, y) \in \mathcal{X} \times \{-1, 1\}$ is:

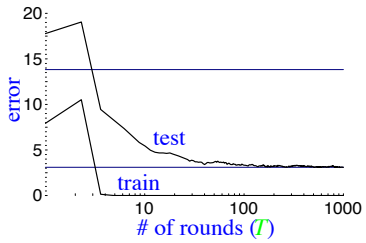
$$\text{(fraction of votes correct)} - \text{(fraction incorrect)} \;=\; yf(x) \in [-1, 1].$$

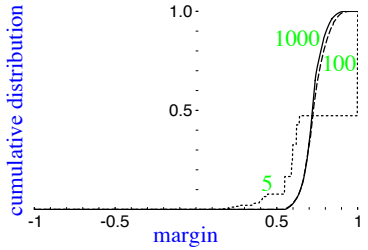
Did H just barely get it right? Or definitively?

- Intuitively: the larger a classifier's margins on the training data, the better its generalization – that is, the lower its true error.
- There are mathematical results that make this precise.
- Adaboost seems to increase the margins on the training points even after training error has gone to zero.

Example revisited

Look at cumulative distribution of margins of training examples:





	# rounds		
	5	100	1000
train error	0.0	0.0	0.0
test error	8.4	3.3	3.1
% margins ≤ 0.5	7.7	0.0	0.0
minimum margin	0.14	0.52	0.55

Another view of boosting

Let $\mathcal{H}$ denote the set of base classifiers $\mathcal{X} \to \{-1, 1\}$.
For instance, $\mathcal{H} = \{\text{decision stumps}\}$.

Imagine a representation of $x \in \mathcal{X}$ in which each $h \in \mathcal{H}$ corresponds to a feature:

$$\Phi(x) = (h(x) : h \in \mathcal{H})$$

The final classifier found by boosting,

$$H(x) = \text{sign}\underbrace{\left(\sum_t \alpha_t h_t(x)\right)}_{\text{call this } f(x)}.$$

is a linear classifier in this enhanced space.

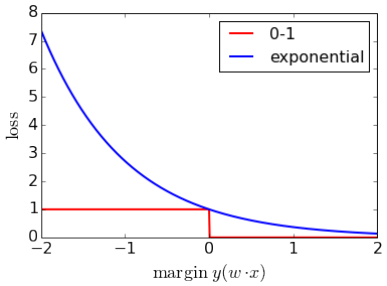
What kind of linear classifier does boosting return? Is it optimizing some loss function?

Minimizing exponential loss

It can be shown that boosting is looking for the linear classifier that minimizes the **exponential loss**:

$$\frac{1}{n}\sum_{i=1}^{n} e^{-y^{(i)}f(x^{(i)})}.$$

This loss function is yet another convex upper bound on 0-1 loss:



Boosting is a **coordinate descent** procedure for minimizing the loss.

Other ensemble methods

1. Endless variants of boosting
2. Bagging (bootstrapping aggregation)
3. Random forests

Bagging

Given a data set S of n labeled points:

- For $t = 1$ to T:
  - Choose n' points randomly, with replacement, from S.
  - Fit a classifier h_t to these points.

(For instance: $T = 100$ or 1000, $n' = n$.)

Final predictor: majority vote of $h_1, \ldots, h_T$.

The resampling and averaging reduces overfitting.

Random forests

Rather like bagging decision trees, but with an additional source of randomization.

Given a data set S of n labeled points:

- For $t = 1$ to T:
  - Choose n' points randomly, with replacement, from S.
  - Fit a decision tree h_t to these points. When building the tree, at each node restrict the split direction to be one of k features chosen at random.

(For instance: $k = \sqrt{p}$ when data is p-dimensional.)

Final predictor: majority vote of $h_1, \ldots, h_T$.

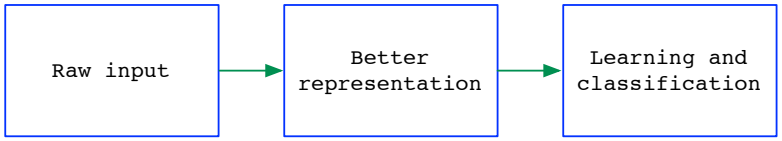
Clustering

CSE 250B

Roadmap for the course

1. Nonparametric methods
   - Nearest neighbor
   - Decision tree
   - Statistical learning theory setup
2. Classification with parametrized models
   - Generative models: product distributions, multinomials, Gaussians
   - Discriminative models: logistic regression
   - Background in linear algebra and optimization
   - More linear classifiers: perceptrons and support vector machines
   - Kernels
3. Combining simple classifiers
   - Boosting, bagging, and random forests
4. **Representation learning**

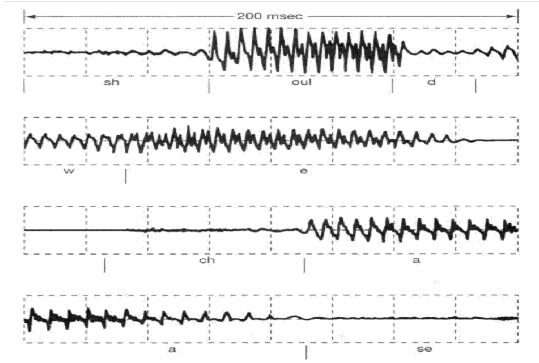
Representation learning



Good representations make learning easier.

- They bring out the true degrees of freedom in the data.
- They capture relevant structure at multiple scales.
- They screen out noisy or irrelevant structure.

Degrees of freedom



Usual representation of speech:

- Take overlapping windows of the speech signal
- Apply many filters within each window
- More filters $\Rightarrow$ higher dimensional

But the speech is produced by a physical system (vocal tract) with a fixed number of degrees of freedom. And the phoneme being uttered can be characterized by the configuration of this apparatus.

Multiscale structure

Commonly-occurring structure at many levels:

- Low-level: like local edges
- Higher-level: like wheels, windows

Representation learning: goals

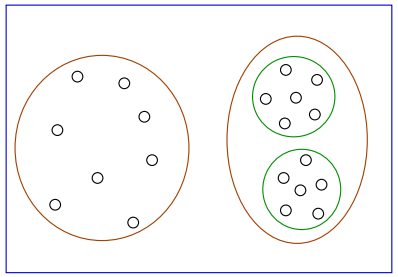
How, and to what extent, can underlying degrees of freedom and multiscale structure be learned from the statistics of unlabeled data?

And when labels are available, how can a good representation be learned in tandem with the classifier?

Topics:

- Clustering
- Informative linear projections
- Embedding and manifold learning
- Metric learning
- Autoencoders
- Deep nets

Clustering in $\mathbb{R}^p$



Two common uses of clustering:

- Vector quantization
  Find a finite set of representatives that provides good coverage of a complex, possibly infinite, high-dimensional space.
- Finding meaningful structure in data
  Finding salient grouping in data.

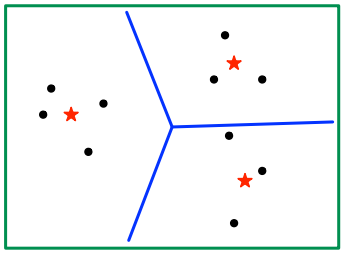
Widely-used clustering methods

1. K-means and its many variants
2. EM for mixtures of Gaussians
3. Agglomerative hierarchical clustering

The k-means optimization problem

- Input: Points $x_1, \ldots, x_n \in \mathbb{R}^p$; integer k
- Output: "Centers", or representatives, $\mu_1, \ldots, \mu_k \in \mathbb{R}^p$
- Goal: Minimize average squared distance between points and their nearest representatives:

$$\text{cost}(\mu_1, \ldots, \mu_k) = \sum_{i=1}^{n} \min_j \|x_i - \mu_j\|^2$$

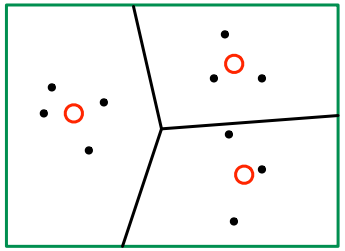


The centers carve $\mathbb{R}^p$ up into k convex regions: μ_j's region consists of points for which it is the closest center.

Lloyd's k-means algorithm

The k-means problem is NP-hard to solve. The most popular heuristic is called the "k-means algorithm".

- Initialize centers $\mu_1, \ldots, \mu_k$ in some manner.
- Repeat until convergence:
  - Assign each point to its closest center.
  - Update each μ_j to the mean of the points assigned to it.



Each iteration reduces the cost $\Rightarrow$ convergence to a local optimum.

Initializing the k-means algorithm

Typical practice: choose k data points at random as the initial centers.

Another common trick: start with extra centers, then prune later.

A particularly good initializer: **k-means++**

- Pick a data point x at random as the first center
- Let $C = \{x\}$ (centers chosen so far)
- Repeat until desired number of centers is attained:
  - Pick a data point x at random from the following distribution:

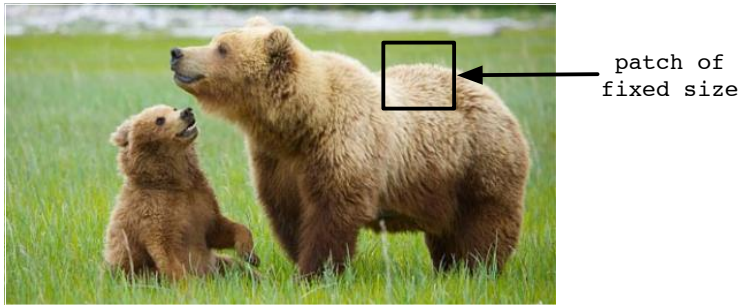
$$\Pr(x) \propto \text{dist}(x, C)^2,$$

where $\text{dist}(x, C) = \min_{z \in C} \|x - z\|$

- Add x to C

Representing images using k-means codewords

Given a collection of images, how to represent as fixed-length vectors?



- Look at all $\ell \times \ell$ patches in all images. Extract features for each.
- Run k-means on this entire collection to get k centers.
- Now associate any image patch with its nearest center.
- Represent an image by a histogram over $\{1, 2, \ldots, k\}$.

Such data sets are truly enormous.

Streaming and online computation

Streaming computation: for data sets that are too large to fit in memory.

- Make one pass (or maybe a few passes) through the data.
- On each pass:
  - See data points one at a time, in order.
  - Update models/parameters along the way.
- There is only enough space to store a tiny fraction of the data, or a perhaps short summary.

Online computation: an even more lightweight setup, for data that is continuously being collected.

- Initialize a model.
- Repeat forever:
  - See a new data point.
  - Update model if need be.

Example: sequential k-means

1. Set the centers $\mu_1, \ldots, \mu_k$ to the first k data points
2. Set their counts to $n_1 = n_2 = \cdots = n_k = 1$
3. Repeat, possibly forever:
   - Get next data point x
   - Let μ_j be the center closest to x
   - Update μ_j and n_j:

$$\mu_j = \frac{n_j \mu_j + x}{n_j + 1} \quad \text{and} \quad n_j = n_j + 1$$

K-means: the good and the bad

The good:

- Fast and easy.
- Effective in quantization.

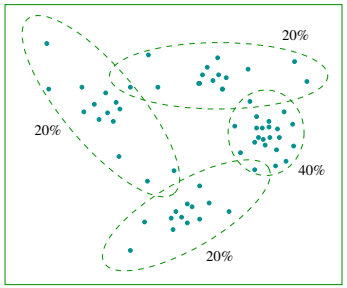
The bad:

- Geared towards data in which the clusters are spherical, and of roughly the same radius.

Is there is a similarly-simple algorithm in which clusters of more general shape are accommodated?

Mixtures of Gaussians

Idea: model each cluster by a Gaussian:



Each of the k clusters is specified by:

- a Gaussian distribution $P_j = N(\mu_j, \Sigma_j)$
- a mixing weight π_j

Overall distribution over $\mathbb{R}^p$: a **mixture of Gaussians**

$$\Pr(x) = \pi_1 P_1(x) + \cdots + \pi_k P_k(x)$$

The clustering task

Given data $x_1, \ldots, x_n \in \mathbb{R}^P$, find the maximum-likelihood mixture of Gaussians: that is, find parameters

- $\pi_1, \ldots, \pi_k \geq 0$ summing to one
- $\mu_1, \ldots, \mu_k \in \mathbb{R}^p$
- $\Sigma_1, \ldots, \Sigma_k \in \mathbb{R}^{p \times p}$

to maximize

$$\begin{aligned}
&\Pr(\text{data} \mid \pi_1 P_1 + \cdots + \pi_k P_k) \\
&= \prod_{i=1}^{n} \left(\sum_{j=1}^{k} \pi_j P_j(x_i) \right) \\
&= \prod_{i=1}^{n} \left(\sum_{j=1}^{k} \frac{\pi_j}{(2\pi)^{p/2} |\Sigma_j|^{1/2}} \exp\left(-\frac{1}{2} (x_i - \mu_j)^T \Sigma_j^{-1} (x_i - \mu_j) \right) \right)
\end{aligned}$$

where P_j is the distribution of the jth cluster, $N(\mu_j, \Sigma_j)$.

The EM algorithm

1. Initialize $\pi_1, \ldots, \pi_k$ and $P_1 = N(\mu_1, \Sigma_1), \ldots, P_k = N(\mu_k, \Sigma_k)$ in some manner.
2. Repeat until convergence:
   - Assign each point x_i fractionally between the k clusters:

$$w_{ij} = \Pr(\text{cluster } j \mid x_i) = \frac{\pi_j P_j(x_i)}{\sum_\ell \pi_\ell P_\ell(x_i)}$$

   - Now update the mixing weights, means, and covariances:

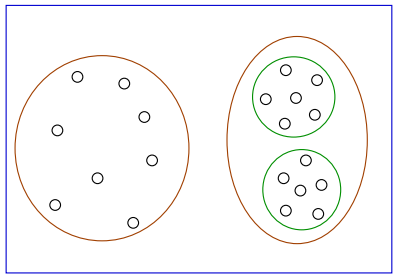
$$\pi_j = \frac{1}{n} \sum_{i=1}^{n} w_{ij}$$

$$\mu_j = \frac{1}{n\pi_j} \sum_{i=1}^{n} w_{ij} x_i$$

$$\Sigma_j = \frac{1}{n\pi_j} \sum_{i=1}^{n} w_{ij} (x_i - \mu_j)(x_i - \mu_j)^T$$

Hierarchical clustering

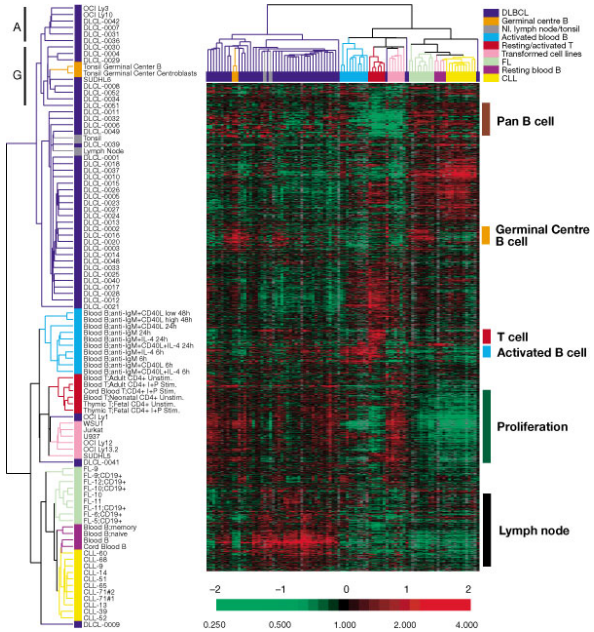
Choosing the number of clusters (k) is difficult.



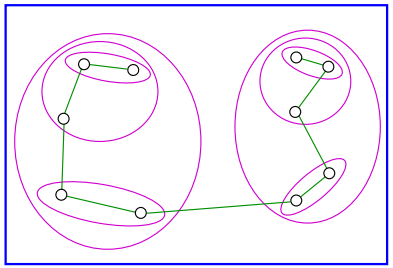
Often there is no single right answer, because of multiscale structure.

Hierarchical clustering avoids these problems.

Example: gene expression data



The single linkage algorithm

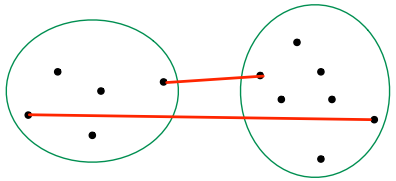


- Start with each point in its own, singleton, cluster
- Repeat until there is just one cluster:
  - Merge the two clusters with the closest pair of points
- Disregard singleton clusters

Linkage methods

- Start with each point in its own, singleton, cluster
- Repeat until there is just one cluster:
  - Merge the two "closest" clusters

How to measure the distance between two clusters of points, C and C'?



- Single linkage

$$\text{dist}(C, C') = \min_{x \in C, x' \in C'} \|x - x'\|$$

- Complete linkage

$$\text{dist}(C, C') = \max_{x \in C, x' \in C'} \|x - x'\|$$

Average linkage

Three commonly-used variants:

1. Average pairwise distance between points in the two clusters

$$\text{dist}(C, C') = \frac{1}{|C| \cdot |C'|} \sum_{x \in C} \sum_{x' \in C'} \|x - x'\|$$

2. Distance between cluster centers

$$\text{dist}(C, C') = \|\text{mean}(C) - \text{mean}(C')\|$$

3. Ward's method: the increase in k-means cost occasioned by merging the two clusters

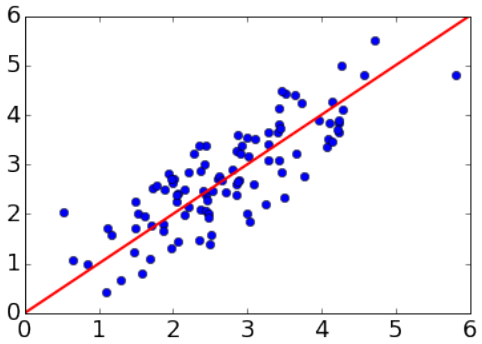
$$\text{dist}(C, C') = \frac{|C| \cdot |C'|}{|C| + |C'|} \|\text{mean}(C) - \text{mean}(C')\|^2$$

Informative projections

CSE 250B

Informative projection

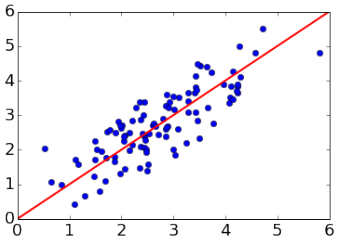
Suppose we wanted just one feature for the following data.



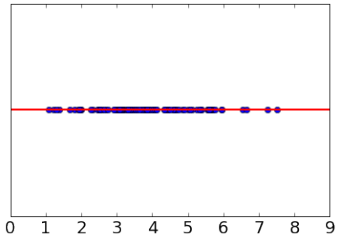
- We could pick a single coordinate.
- Or an arbitrary direction.

A good choice: the **direction of maximum variance**.

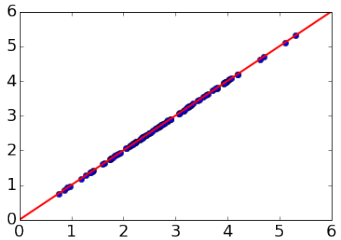
Two types of projection



Projection onto $\mathbb{R}$:

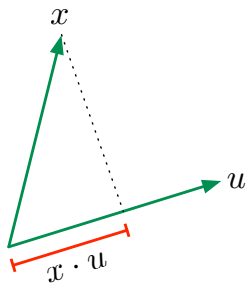


Projection onto a 1-d line in $\mathbb{R}^2$:



Projection: formally

What is the projection of $x \in \mathbb{R}^p$ onto direction $u \in \mathbb{R}^p$ (where $\|u\| = 1$)?



As a one-dimensional value:

$$x \cdot u = u \cdot x = u^T x = \sum_{i=1}^{p} u_i x_i.$$

As a p-dimensional vector:

$$(x \cdot u)u = uu^T x$$

"Move $x \cdot u$ units in direction u"

What is the projection of $x = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$ onto the following directions?

- The coordinate direction e_1? Answer: 2
- The direction $\begin{pmatrix} 1 \\ -1 \end{pmatrix}$? Answer: $-1/\sqrt{2}$

Projection onto multiple directions

Want to project $x \in \mathbb{R}^p$ into the k-dimensional subspace defined by vectors $u_1, \ldots, u_k \in \mathbb{R}^p$.

This is easiest when the u_i's are **orthonormal**:

- They each have length one.
- They are at right angles to each other: $u_i \cdot u_j = 0$ whenever $i \neq j$

Then the projection, as a k-dimensional vector, is

$$(x \cdot u_1, x \cdot u_2, \ldots, x \cdot u_k) = \underbrace{\begin{pmatrix} \longleftarrow & u_1 & \longrightarrow \\ \longleftarrow & u_2 & \longrightarrow \\ & \vdots & \\ \longleftarrow & u_k & \longrightarrow \end{pmatrix}}_{\text{call this } U^T} \begin{pmatrix} \uparrow \\ x \\ \downarrow \end{pmatrix}$$

As a p-dimensional vector, the projection is

$$(x \cdot u_1)u_1 + (x \cdot u_2)u_2 + \cdots + (x \cdot u_k)u_k = UU^T x.$$

Projection onto multiple directions: example

Suppose data are in $\mathbb{R}^4$ and we want to project onto the first two coordinates.

Take vectors $u_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$, $u_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$ (notice: orthonormal)

Then write $U^T = \begin{pmatrix} \longleftarrow & u_1 & \longrightarrow \\ \longleftarrow & u_2 & \longrightarrow \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$

The projection of $x \in \mathbb{R}^4$, as a 2-d vector, is

$$U^T x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

The projection of x as a 4-d vector is

$$UU^T x = \begin{pmatrix} x_1 \\ x_2 \\ 0 \\ 0 \end{pmatrix}$$

But we'll generally project along non-coordinate directions.

The best single direction

Suppose we need to map our data $x \in \mathbb{R}^p$ into just **one** dimension:

$$x \mapsto u \cdot x \quad \text{for some unit direction } u \in \mathbb{R}^p$$

What is the direction u of maximum variance?

Theorem: Let Σ be the $p \times p$ covariance matrix of X. The variance of X in direction u is given by $u^T \Sigma u$.

- Suppose the mean of X is $\mu \in \mathbb{R}^p$. The projection $u^T X$ has mean

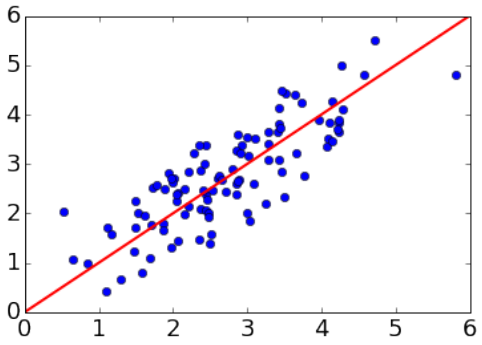
$$\mathbb{E}(u^T X) = u^T \mathbb{E}X = u^T \mu.$$

- The variance of $u^T X$ is

$$\begin{aligned} \text{var}(u^T X) &= \mathbb{E}(u^T X - u^T \mu)^2 = \mathbb{E}(u^T (X - \mu)(X - \mu)^T u) \\ &= u^T \mathbb{E}(X - \mu)(X - \mu)^T u = u^T \Sigma u. \end{aligned}$$

Another theorem: $u^T \Sigma u$ is maximized by setting u to the first **eigenvector** of Σ. The maximum value is the corresponding **eigenvalue**.

Best single direction: example



This direction is the **first eigenvector** of the 2×2 covariance matrix of the data.

The best k-dimensional projection

Let Σ be the $p \times p$ covariance matrix of X. Its **eigendecomposition** can be computed in $O(p^3)$ time and consists of:

- real **eigenvalues** $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p$
- corresponding **eigenvectors** $u_1, \ldots, u_p \in \mathbb{R}^p$ that are orthonormal: that is, each u_i has unit length and $u_i \cdot u_j = 0$ whenever $i \neq j$.

Theorem: Suppose we want to map data $X \in \mathbb{R}^p$ to just k dimensions, while capturing as much of the variance of X as possible. The best choice of projection is:

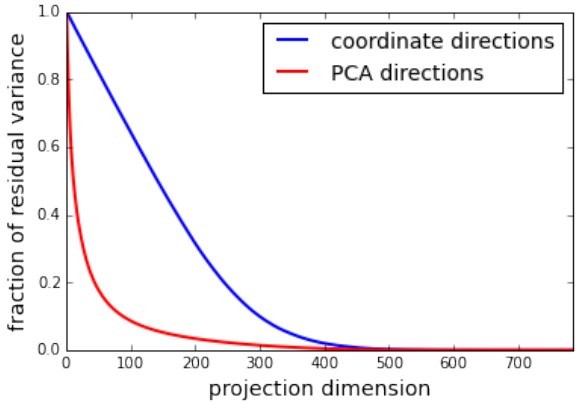
$$x \mapsto (u_1 \cdot x, u_2 \cdot x, \ldots, u_k \cdot x),$$

where u_i are the eigenvectors described above.

Projecting the data in this way is **principal component analysis** (PCA).

Example: MNIST

Contrast coordinate projections with PCA:



MNIST: image reconstruction

Reconstruct this original image from its PCA projection to k dimensions.

$k = 200$ $k = 150$ $k = 100$ $k = 50$

Q: What are these reconstructions exactly?
A: Image x is reconstructed as $UU^T x$, where U is a $p \times k$ matrix whose columns are the top k eigenvectors of Σ.

What are eigenvalues and eigenvectors?

There are several steps to understanding these.

1. Any matrix M defines a function (or **transformation**) $x \mapsto Mx$.
2. If M is a $p \times q$ matrix, then this transformation maps vector $x \in \mathbb{R}^q$ to vector $Mx \in \mathbb{R}^p$.
3. We call it a **linear transformation** because $M(x + x') = Mx + Mx'$.
4. We'd like to understand the nature of these transformations. The easiest case is when M is **diagonal**:

$$\underbrace{\begin{pmatrix} 2 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 10 \end{pmatrix}}_{M} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} 2x_1 \\ -x_2 \\ 10x_3 \end{pmatrix}}_{Mx}$$

   In this case, M simply scales each coordinate separately.
5. What about more general matrices that are symmetric but not necessarily diagonal? They also just scale coordinates separately, but in a **different coordinate system**.

Eigenvalue and eigenvector: definition

Let M be a $p \times p$ matrix.
We say $u \in \mathbb{R}^p$ is an **eigenvector** if M maps u onto the same direction, that is,

$$Mu = \lambda u$$

for some scaling constant λ. This λ is the **eigenvalue** associated with u.

Question: What are the eigenvectors and eigenvalues of:

$$M = \begin{pmatrix} 2 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 10 \end{pmatrix} ?$$

Answer: Eigenvectors e_1, e_2, e_3, with corresponding eigenvalues $2, -1, 10$.

Notice that these eigenvectors form an orthonormal basis.

Eigenvectors of a real symmetric matrix

Theorem. Let M be any real symmetric $p \times p$ matrix. Then M has

- p eigenvalues $\lambda_1, \ldots, \lambda_p$
- corresponding eigenvectors $u_1, \ldots, u_p \in \mathbb{R}^p$ that are orthonormal

We can think of $u_1, \ldots, u_p$ as being the axes of the natural coordinate system for understanding M.

Example: consider the matrix

$$M = \begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}$$

It has eigenvectors

$$u_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad u_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

and corresponding eigenvalues $\lambda_1 = 4$ and $\lambda_2 = 2$. (Check)

Spectral decomposition

Theorem. Let M be any real symmetric $p \times p$ matrix. Then M has

- p eigenvalues $\lambda_1, \ldots, \lambda_p$
- corresponding eigenvectors $u_1, \ldots, u_p \in \mathbb{R}^p$ that are orthonormal

Spectral decomposition: Here is another way to write M:

$$M = \underbrace{\begin{pmatrix} \uparrow & \uparrow & & \uparrow \\ u_1 & u_2 & \cdots & u_p \\ \downarrow & \downarrow & & \downarrow \end{pmatrix}}_{U\text{: columns are eigenvectors}} \underbrace{\begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_p \end{pmatrix}}_{\Lambda\text{: eigenvalues on diagonal}} \underbrace{\begin{pmatrix} \longleftarrow & u_1 & \longrightarrow \\ \longleftarrow & u_2 & \longrightarrow \\ & \vdots & \\ \longleftarrow & u_p & \longrightarrow \end{pmatrix}}_{U^T}$$

Thus $Mx = U\Lambda U^T x$, which can be interpreted as follows:

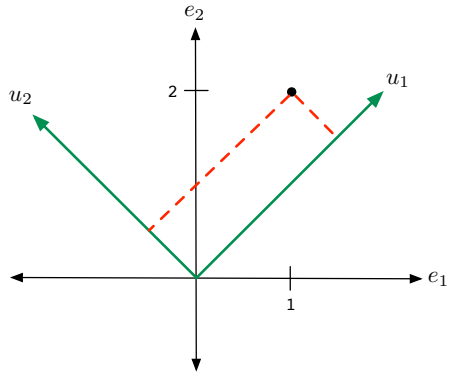
- U^T rewrites x in the $\{u_i\}$ coordinate system
- Λ is a simple coordinate scaling in that basis
- U then sends the scaled vector back into the usual coordinate basis

Spectral decomposition: example

Apply spectral decomposition to the matrix M we saw earlier:

$$M = \begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix} = \underbrace{\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}}_{U} \underbrace{\begin{pmatrix} 4 & 0 \\ 0 & 2 \end{pmatrix}}_{\Lambda} \underbrace{\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}}_{U^T}$$

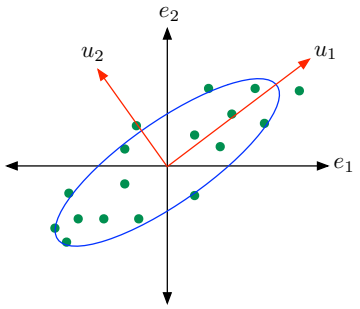
$$\begin{aligned} M \begin{pmatrix} 1 \\ 2 \end{pmatrix} &= ??? \\ &= U\Lambda U^T \begin{pmatrix} 1 \\ 2 \end{pmatrix} \\ &= U\Lambda \frac{1}{\sqrt{2}} \begin{pmatrix} 3 \\ 1 \end{pmatrix} \\ &= U \frac{1}{\sqrt{2}} \begin{pmatrix} 12 \\ 2 \end{pmatrix} \\ &= \begin{pmatrix} 5 \\ 7 \end{pmatrix} \end{aligned}$$



Principal component analysis: recap

Consider data vectors $X \in \mathbb{R}^p$.

- The covariance matrix Σ is a $p \times p$ symmetric matrix.
- Get eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p$, eigenvectors $u_1, \ldots, u_p$.
- $u_1, \ldots, u_p$ is an alternative basis in which to represent the data.
- The variance of X in direction u_i is λ_i.
- To project to k dimensions while losing as little as possible of the overall variance, use $x \mapsto (x \cdot u_1, \ldots, x \cdot u_k)$.



What is the covariance of the projected data?

Example: personality assessment

What are the dimensions along which personalities differ?

- *Lexical hypothesis*: most important personality characteristics have become encoded in natural language.
- Allport and Odbert (1936): sat down with the English dictionary and extracted all terms that could be used to distinguish one person's behavior from another's. Roughly 18000 words, of which 4500 could be described as personality traits.
- Step: group these words into (approximate) synonyms. This is done by manual clustering. e.g. Norman (1967):

Spirit	Jolly, merry, witty, lively, peppy
Talkativeness	Talkative, articulate, verbose, gossipy
Sociability	Companionable, social, outgoing
Spontaneity	Impulsive, carefree, playful, zany
Boisterousness	Mischievous, rowdy, loud, prankish
Adventure	Brave, venturous, fearless, reckless
Energy	Active, assertive, dominant, energetic
Conceit	Boastful, conceited, egotistical
Vanity	Affected, vain, chic, dapper, jaunty
Indiscretion	Nosey, snoopy, indiscreet, meddlesome
Sensuality	Sexy, passionate, sensual, flirtatious

- Data collection: Ask a variety of subjects to what extent each of these words describes them.

Personality assessment: the data

Matrix of data (1 = strongly disagree, 5 = strongly agree)

	shy	merry	tense	boastful	forgiving	quiet
Person 1	4	1	1	2	5	5
Person 2	1	4	4	5	2	1
Person 3	2	4	5	4	2	2
		⋮				

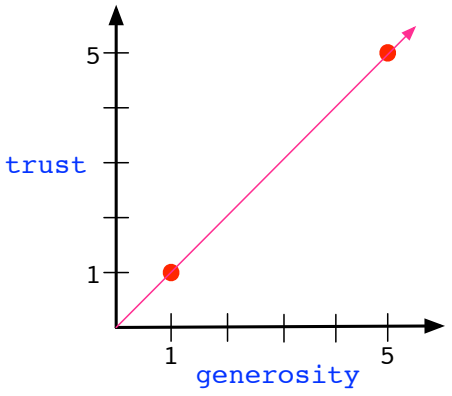
How to extract important directions?

- Treat each column as a data point, find tight clusters
- Treat each row as a data point, apply PCA
- Other ideas: factor analysis, independent component analysis, ...

Many of these yield similar results

What does PCA accomplish?

Example: suppose two traits (generosity, trust) are highly correlated, to the point where each person either answers "1" to both or "5" to both.



This single PCA dimension entirely accounts for the two traits.

The "Big Five" taxonomy

Extraversion		Agreeableness		Conscientiousness		Neuroticism		Oppenness/Intellect	
Low	High	Low	High	Low	High	Low	High	Low	High
-.83 Quiet	.85 Talkative	-.52 Fault-finding	.87 Sympathetic	-.58 Careless	.80 Organized	-.39 Stable*	.73 Tense	-.74 Commonplace	.76 Wide interests
-.80 Reserved	.83 Assertive	-.48 Cold	.85 Kind	-.53 Disorderly	.80 Thorough	-.35 Calm*	.72 Anxious	-.73 Narrow interests	.76 Imaginative
-.75 Shy	.82 Active	-.45 Unfriendly	.85 Appreciative	-.50 Frivolous	.78 Planful	-.21 Contented*	.72 Nervous	-.67 Simple	.72 Intelligent
-.71 Silent	.82 Energetic	-.45 Quarrelsome	.84 Affectionate	-.49 Irresponsible	.78 Efficient	.14 Unemotional*	.71 Moody	-.55 Shallow	.73 Original
-.67 Withdrawn	.82 Outgoing	-.45 Hard-hearted	.84 Soft-hearted	-.40 Slipshot	.73 Responsible		.71 Worrying	-.47 Unintelligent	.68 Insightful
-.66 Retiring	.80 Outspoken	-.38 Unkind	.82 Warm	-.39 Undependable	.72 Reliable		.68 Touchy		.64 Curious
	.79 Dominant	-.33 Cruel	.81 Generous	-.37 Forgetful	.70 Dependable		.64 Fearful		.59 Sophisticated
	.73 Forceful	-.31 Stern*	.78 Trusting		.68 Conscientious		.63 High-strung		.59 Artistic
	.73 Enthusiastic	-.28 Thankless	.77 Helpful		.66 Precise		.63 Self-pitying		.59 Clever
	.68 Show-off	-.24 Stingy*	.77 Forgiving		.66 Practical		.60 Temperamental		.58 Inventive
	.68 Sociable		.74 Pleasant		.65 Deliberate		.59 Unstable		.56 Sharp-witted
	.64 Spunky		.73 Good-natured		.46 Painstaking		.58 Self-punishing		.55 Ingenious
	.64 Adventurous		.73 Friendly		.26 Cautious*		.54 Despondent		.45 Witty*
	.62 Noisy		.72 Cooperative				.51 Emotional		.45 Resourceful*
	.58 Bossy		.67 Gentle						.37 Wise
			.66 Unselfish						.33 Logical*
			.56 Praising						.29 Civilized*
			.51 Sensitive						.22 Foresighted*
									.21 Polished*
									.20 Dignified*

Many applications, such as online match-making.

Singular value decomposition (SVD)

For **symmetric** matrices, such as covariance matrices, we have seen:

- Results about existence of eigenvalues and eigenvectors
- The fact that the eigenvectors form an alternative basis
- The resulting spectral decomposition, which is used in PCA

But what about arbitrary matrices $M \in \mathbb{R}^{p \times q}$?

Any $p \times q$ matrix (say $p \leq q$) has a **singular value decomposition**:

$$M = \underbrace{\begin{pmatrix} \uparrow & & \uparrow \\ u_1 & \cdots & u_p \\ \downarrow & & \downarrow \end{pmatrix}}_{p \times p \text{ matrix } U} \underbrace{\begin{pmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_p \end{pmatrix}}_{p \times p \text{ matrix } \Lambda} \underbrace{\begin{pmatrix} \longleftarrow & v_1 & \longrightarrow \\ & \vdots & \\ \longleftarrow & v_p & \longrightarrow \end{pmatrix}}_{p \times q \text{ matrix } V^T}$$

- $u_1, \ldots, u_p$ are orthonormal vectors in $\mathbb{R}^p$
- $v_1, \ldots, v_p$ are orthonormal vectors in $\mathbb{R}^q$
- $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p$ are **singular values**

Matrix approximation

We can **factor** any $p \times q$ matrix as $M = UW^T$:

$$M = \begin{pmatrix} \uparrow & & \uparrow \\ u_1 & \cdots & u_p \\ \downarrow & & \downarrow \end{pmatrix} \begin{pmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_p \end{pmatrix} \begin{pmatrix} \longleftarrow & v_1 & \longrightarrow \\ & \vdots & \\ \longleftarrow & v_p & \longrightarrow \end{pmatrix}$$

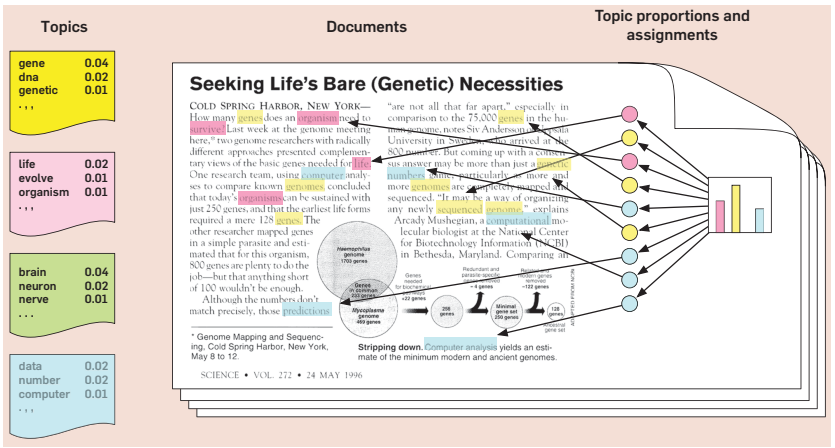
$$= \underbrace{\begin{pmatrix} \uparrow & & \uparrow \\ u_1 & \cdots & u_p \\ \downarrow & & \downarrow \end{pmatrix}}_{p \times p \text{ matrix } U} \underbrace{\begin{pmatrix} \longleftarrow & \sigma_1 v_1 & \longrightarrow \\ & \vdots & \\ \longleftarrow & \sigma_p v_p & \longrightarrow \end{pmatrix}}_{p \times q \text{ matrix } W^T}$$

A concise approximation to M: just take the first k columns of U and the first k rows of W^T, for $k < p$:

$$\widehat{M} = \underbrace{\begin{pmatrix} \uparrow & & \uparrow \\ u_1 & \cdots & u_k \\ \downarrow & & \downarrow \end{pmatrix}}_{p \times k} \underbrace{\begin{pmatrix} \longleftarrow & \sigma_1 v_1 & \longrightarrow \\ & \vdots & \\ \longleftarrow & \sigma_k v_k & \longrightarrow \end{pmatrix}}_{k \times q}$$

Example: topic modeling

Blei (2012):



Latent semantic indexing (LSI)

Given a large corpus of n documents:

- Fix a vocabulary, say of V words.
- Bag-of-words representation for documents: each document becomes a vector of length V, with one coordinate per word.
- The corpus is an $n \times V$ matrix, one row per document.

	cat	dog	house	boat	garden	...
Doc 1	4	1	1	0	2	
Doc 2	0	0	3	1	0	
Doc 3	0	1	3	0	0	
		⋮				

Let's find a concise approximation to this matrix M.

Latent semantic indexing, cont'd

Use SVD to get an approximation to M: for small k,

$$\underbrace{\begin{pmatrix} \longleftarrow \text{doc } 1 \longrightarrow \\ \longleftarrow \text{doc } 2 \longrightarrow \\ \longleftarrow \text{doc } 3 \longrightarrow \\ \vdots \\ \longleftarrow \text{doc } n \longrightarrow \end{pmatrix}}_{n \times V \text{ matrix } M} \approx \underbrace{\begin{pmatrix} \longleftarrow \theta_1 \longrightarrow \\ \longleftarrow \theta_2 \longrightarrow \\ \longleftarrow \theta_3 \longrightarrow \\ \vdots \\ \longleftarrow \theta_n \longrightarrow \end{pmatrix}}_{n \times k \text{ matrix } \Theta} \underbrace{\begin{pmatrix} \longleftarrow \Psi_1 \longrightarrow \\ \vdots \\ \longleftarrow \Psi_k \longrightarrow \end{pmatrix}}_{k \times V \text{ matrix } \Psi}$$

Think of this as a *topic model* with k topics.

- Ψ_j is a vector of length V describing topic j: coefficient Ψ_{jw} is large if word w appears often in that topic.
- Each document is a combination of topics: θ_{ij} is the weight of topic j in document i.

Document i originally represented by ith row of M, a vector in $\mathbb{R}^V$.
Can instead use $\theta_i \in \mathbb{R}^k$, a more concise "semantic" representation.

The rank of a matrix

Suppose we want to approximate a matrix M by a simpler matrix $\widehat{M}$.
What is a suitable notion of "simple"?

- Let's say M and $\widehat{M}$ are $p \times q$, where $p \leq q$.
- Treat each row of $\widehat{M}$ as a data point in $\mathbb{R}^q$.
- We can think of the data as "simple" if it actually lies in a low-dimensional subspace.
- If the rows lie in k-dimensional subspace, we say that $\widehat{M}$ has **rank** k.

The **rank** of a matrix is the number of linearly independent rows.

Low-rank approximation: given $M \in \mathbb{R}^{p \times q}$ and an integer k, find the matrix $\widehat{M} \in \mathbb{R}^{p \times q}$ that is the best rank-k approximation to M.

That is, find $\widehat{M}$ so that

- $\widehat{M}$ has rank $\leq k$
- The approximation error $\sum_{i,j}(M_{ij} - \widehat{M}_{ij})^2$ is minimized.

We can get $\widehat{M}$ directly from the singular value decomposition of M.

Low-rank approximation

Recall: Singular value decomposition of $p \times q$ matrix M (with $p \leq q$):

$$M = \begin{pmatrix} \uparrow & & \uparrow \\ u_1 & \cdots & u_p \\ \downarrow & & \downarrow \end{pmatrix} \begin{pmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_p \end{pmatrix} \begin{pmatrix} \longleftarrow v_1 \longrightarrow \\ \vdots \\ \longleftarrow v_p \longrightarrow \end{pmatrix}$$

- $u_1, \ldots, u_p$ is an orthonormal basis of $\mathbb{R}^p$
- $v_1, \ldots, v_q$ is an orthonormal basis of $\mathbb{R}^q$
- $\sigma_1 \geq \cdots \geq \sigma_p$ are **singular values**

The **best rank-k approximation** to M, for any $k \leq p$, is then

$$\widehat{M} = \underbrace{\begin{pmatrix} \uparrow & & \uparrow \\ u_1 & \cdots & u_k \\ \downarrow & & \downarrow \end{pmatrix}}_{p \times k} \underbrace{\begin{pmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_k \end{pmatrix}}_{k \times k} \underbrace{\begin{pmatrix} \longleftarrow v_1 \longrightarrow \\ \vdots \\ \longleftarrow v_k \longrightarrow \end{pmatrix}}_{k \times q}$$

Example: Collaborative filtering

Details and images from Koren, Bell, Volinksy (2009).

Recommender systems: matching customers with products.

- Given: data on prior purchases/interests of users
- Recommend: further products of interest

Prototypical example: Netflix.

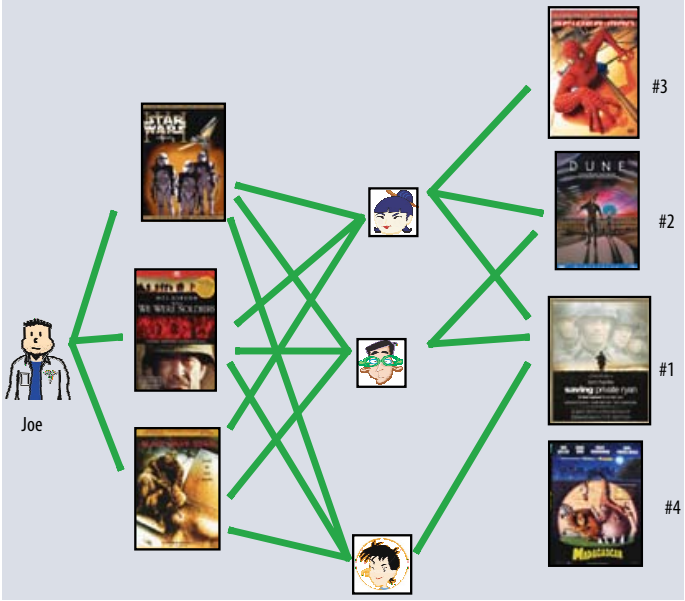
A successful approach: **collaborative filtering**.

- Model dependencies between different products, and between different users.
- Can give reasonable recommendations to a relatively new user.

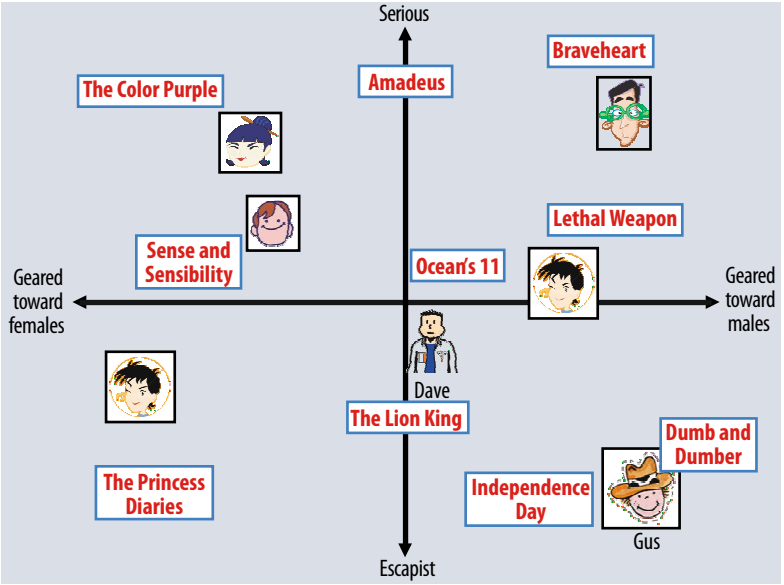
Two strategies for collaborative filtering:

- Neighborhood methods
- Latent factor methods

Neighborhood methods



Latent factor methods



The matrix factorization approach

User ratings are assembled in a large matrix M:

	Star Wars	Matrix	Casablanca	Camelot	Godfather	...
User 1	5	5	2	0	0	
User 2	0	0	3	4	5	
User 3	0	0	5	0	0	
		⋮				

- Not rated = 0, otherwise scores 1-5.
- For n users and p movies, this has size $n \times p$.
- Most of the entries are unavailable, and we'd like to predict these.

Idea: Find the best low-rank approximation of M, and use it to fill in the missing entries.

User and movie factors

Best rank-k approximation is of the form $M \approx UW^T$:

$$\underbrace{\begin{pmatrix} \longleftarrow \text{ user } 1 \longrightarrow \\ \longleftarrow \text{ user } 2 \longrightarrow \\ \longleftarrow \text{ user } 3 \longrightarrow \\ \vdots \\ \longleftarrow \text{ user } n \longrightarrow \end{pmatrix}}_{n \times p \text{ matrix } M} \approx \underbrace{\begin{pmatrix} \longleftarrow u_1 \longrightarrow \\ \longleftarrow u_2 \longrightarrow \\ \longleftarrow u_3 \longrightarrow \\ \vdots \\ \longleftarrow u_n \longrightarrow \end{pmatrix}}_{n \times k \text{ matrix } U} \underbrace{\begin{pmatrix} \uparrow & \uparrow & & \uparrow \\ w_1 & w_2 & \cdots & w_p \\ \downarrow & \downarrow & & \downarrow \end{pmatrix}}_{k \times p \text{ matrix } W^T}$$

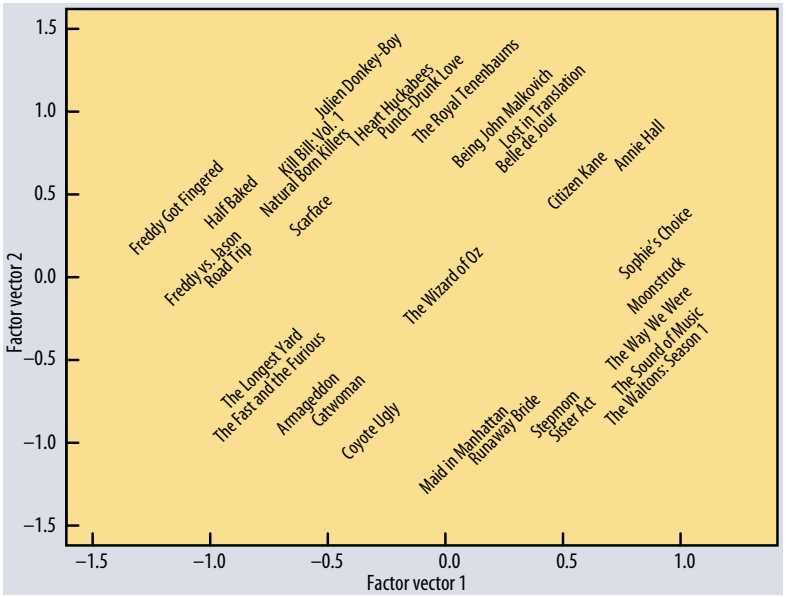
Thus user i's rating of movie j is approximated as

$$M_{ij} \approx u_i \cdot w_j$$

This "latent" representation embeds users and movies within the same k-dimensional space:

- Represent ith user by $u_i \in \mathbb{R}^k$
- Represent jth movie by $w_j \in \mathbb{R}^k$

Top two Netflix factors



Beyond projections

CSE 250B

Beyond projections

PCA and SVD find informative linear projections. Given a data set in $\mathbb{R}^p$, and a number $k < p$, they:

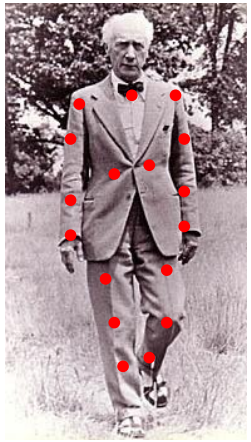
- Find orthogonal directions $u_1, \ldots, u_k \in \mathbb{R}^p$
- Approximate points in $\mathbb{R}^p$ by their projection into the subspace spanned by these directions

Two ways in which we'd like to generalize this.

1. Manifold learning
   What if the data lies on (or near) a nonlinear surface?
2. Dictionary learning
   What if we want the basis vectors $u_1, \ldots, u_k$ to have other special properties: for instance, that the data points x have a *sparse representation* in terms of these directions?

Low dimensional manifolds

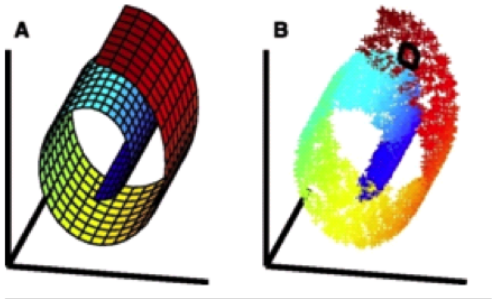
Sometimes data in a high-dimensional space $\mathbb{R}^p$ in fact lies close to a k-dimensional manifold, for $k \ll p$



1. Motion capture
   M markers on a human body yields data in $\mathbb{R}^{3M}$
2. Speech signals
   Representation can be made arbitrarily high dimensional by applying more filters to each window of the time series

This whole area: "Manifold learning"

The ISOMAP algorithm

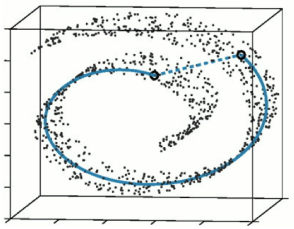


ISOMAP (Tenenbaum et al, 1999): given data $x^{(1)}, \ldots, x^{(n)} \in \mathbb{R}^p$,

1. Estimate *geodesic distances* between the data points: that is, distances along the manifold.
2. Then embed these points into Euclidean space so as to (approximately) match these distances.

How can these two steps be achieved?

Estimating geodesic distances



Key idea: for **nearby** pairs of points, Euclidean distance and geodesic distance are approximately the same.

1. Construct the **neighborhood graph.**
   Given data $x^{(1)}, \ldots, x^{(n)}$, construct a graph $G = (V, E)$ with
   - Nodes $V = \{1, 2, \ldots, n\}$ (one per data point)
   - Edges $(i, j) \in E$ whenever $x^{(i)}$ and $x^{(j)}$ are close together
2. Compute distances in this graph.
   Set the length of any $(i, j) \in E$ to $\|x^{(i)} - x^{(j)}\|$. Compute all pairwise distances between nodes, using a shortest-paths algorithm.

Distance-preserving embeddings

The algorithmic task:

- *Input:* An $n \times n$ matrix of pairwise distances

  $$D_{ij} = \text{desired distance between points } i \text{ and } j,$$

  as well as an integer k.
- *Output:* an embedding $z^{(1)}, \ldots, z^{(n)} \in \mathbb{R}^k$ that realizes these distances as closely as possible.

Most widely-used algorithm: **classical multidimensional scaling.**

- Let $D \in \mathbb{R}^{n \times n}$ be the matrix of desired *squared* interpoint distances.
- Schoenberg (1938): D can be realized in Euclidean space if and only if $B = -\frac{1}{2}HDH$ is positive semidefinite, where $H = I_n - \frac{1}{n}11^T$.
- In fact, looking at this matrix B suggests an embedding even if it is not positive semidefinite.

The Gram matrix

For points in Euclidean space, it is easy to go from dot products to squared interpoint distances.

$$\|x - x'\|^2 \;=\; \|x\|^2 + \|x'\|^2 - 2x \cdot x' \;=\; x \cdot x + x' \cdot x' - 2x \cdot x'.$$

What about going from squared distances to dot products?

Let $z^{(1)}, \ldots, z^{(n)}$ be points in Euclidean space.

- Let $D_{ij} = \|z^{(i)} - z^{(j)}\|^2$ be the squared interpoint distances.
- Let $B_{ij} = z^{(i)} \cdot z^{(j)}$ be the dot products. This is the **Gram matrix**.

Moving between D and B:

- We've seen: $D_{ij} = B_{ii} + B_{jj} - B_{ij} - B_{ji}$. That is, D is linear in B.
- A little algebra also shows that for $H = I_n - \frac{1}{n}11^T$,

$$B = -\frac{1}{2}HDH.$$

The Gram matrix is convenient: we can read off an embedding from it.

Recovering an embedding based only on distances

Let $D \in \mathbb{R}^{n \times n}$ be a matrix of desired *squared* interpoint distances. Suppose these are realizable in Euclidean space: that is, there exist vectors $z^{(1)}, \ldots, z^{(n)}$ such that $D_{ij} = \|z^{(i)} - z^{(j)}\|^2$.

We have seen that we can easily obtain the Gram matrix, $B_{ij} = z^{(i)} \cdot z^{(j)}$.

- B is p.s.d. (why?). Thus its eigenvalues are nonnegative.
- Compute spectral decomposition:

  $$B \;=\; U\Lambda U^T \;=\; YY^T,$$

  where Λ is the diagonal matrix of eigenvalues and $Y = U\Lambda^{1/2}$.
- Denote the rows of Y by $y^{(1)}, \ldots, y^{(n)}$. Then

  $$y^{(i)} \cdot y^{(j)} = (YY^T)_{ij} = B_{ij} = z^{(i)} \cdot z^{(j)}.$$

  If dot products are preserved, so are distances.

Result: an embedding $y^{(1)}, \ldots, y^{(n)}$ that exactly replicates distances D.

What is the dimensionality of this embedding?

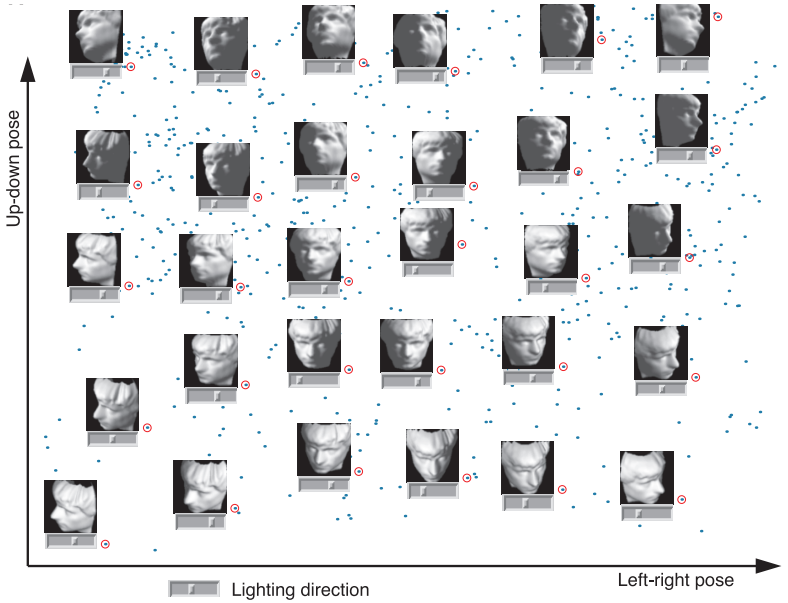
Classical multidimensional scaling

A slight generalization works even when the distances cannot necessarily be realized in Euclidean space.

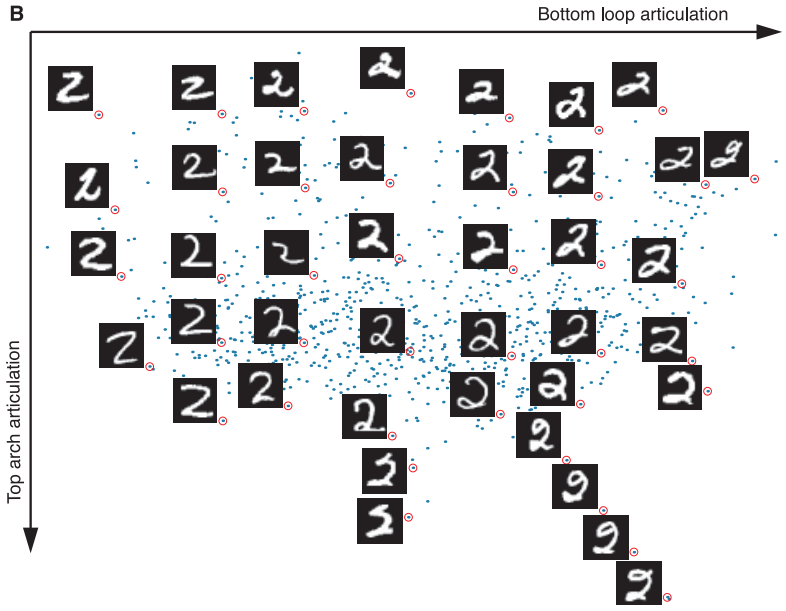
Given $n \times n$ matrix D of squared interpoint distances, and target dimension k:

1. Compute $B = -\frac{1}{2}HDH$.
2. Compute the spectral decomposition $B = U\Lambda U^T$, where the eigenvalues in Λ are arranged in decreasing order.
3. Zero out any negative entries of Λ to get Λ_+.
4. Set $Y = U\Lambda_+^{1/2}$.
5. Set Y_k to the first k columns of Y.
6. Let the embedding of the n points be given by the rows of Y_k.

ISOMAP: examples



ISOMAP: examples



More manifold learning

1. Other good algorithms, such as
   - Locally linear embedding
   - Laplacian eigenmaps
   - Maximum variance unfolding
2. Notions of intrinsic dimensionality
3. Statistical rates of convergence for data lying on manifolds
4. Capturing other kinds of topological structure

Dictionary learning

Given data points $x^{(1)}, \ldots, x^{(n)} \in \mathbb{R}^p$, and an integer m:

- Choose m dictionary vectors $\phi_1, \ldots, \phi_m \in \mathbb{R}^p$.
- Approximate each $x^{(i)}$ by a linear combination of these dictionary elements.

$$\underbrace{\begin{pmatrix} \uparrow & & \uparrow \\ x^{(1)} & \cdots & x^{(n)} \\ \downarrow & & \downarrow \end{pmatrix}}_{\text{data matrix } X} \approx \underbrace{\begin{pmatrix} \uparrow & \uparrow & & \uparrow \\ \phi_1 & \phi_2 & \cdots & \phi_m \\ \downarrow & \downarrow & & \downarrow \end{pmatrix}}_{\text{dictionary } \Phi} \underbrace{\begin{pmatrix} \uparrow & & \uparrow \\ s^{(1)} & \cdots & s^{(n)} \\ \downarrow & & \downarrow \end{pmatrix}}_{\text{encoding } S}$$

- Principal component analysis: the ϕ_i are orthogonal and $m \leq p$
- Independent component analysis: the rows of S are approximately statistically independent and $m \leq p$
- Sparse coding: the columns of S are sparse and often $m > p$ ("overcomplete basis")

Sparse coding

Given $x^{(1)}, \ldots, x^{(n)} \in \mathbb{R}^p$, find dictionary vectors $\phi_1, \ldots, \phi_m$ and sparse representations $s^{(1)}, \ldots, s^{(n)} \in \mathbb{R}^m$ such that

$$x^{(i)} \approx \Phi s^{(i)}.$$

Optimization problem: find matrices Φ, S that minimize

$$\begin{aligned} L(\Phi, S) &= \|X - \Phi S\|_F^2 - \lambda \cdot \text{sparsity}(S) \\ &= \sum_{i=1}^{n} \left(\|x^{(i)} - \Phi s^{(i)}\|^2 - \lambda \cdot \text{sparsity}(s^{(i)}) \right) \end{aligned}$$

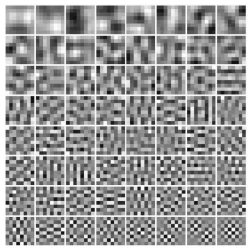
Alternating minimization procedure:

- Initialize Φ somehow
- Repeat until convergence:
  - Fixing Φ, minimize $L(\cdot)$ over S
  - Fixing S, minimize $L(\cdot)$ over Φ

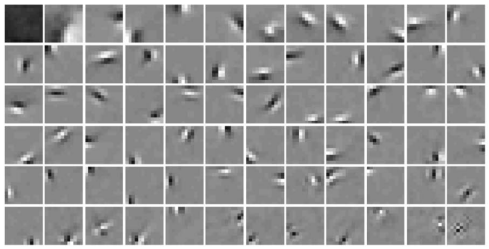
Example: image patches

Olshausen-Field (1996), Lewicki-Olshausen (1999): PCA versus sparse coding for natural image patches.

PCA



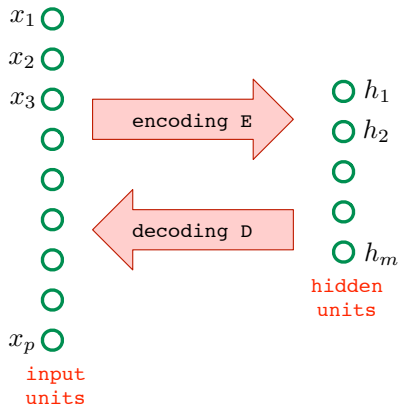
sparse coding



Sparse coding does a much better job at finding a basis that resembles the receptive fields of simple cells in visual cortex.

Autoencoders

A generalization of dictionary learning.

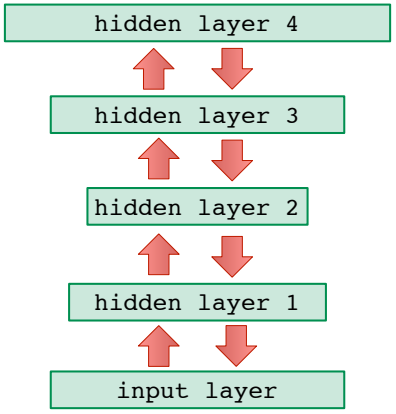


Here E and D might be probabilistic maps. Fit them so that

$$x \approx D(E(x)) \text{ on data points } x \in \mathbb{R}^p.$$

Stacked autoencoders

Successively higher-level representations



One way to fit these models (using unlabeled data):

- Fit one layer at a time to the previous layer's activations
- Then fine-tune the whole structure to minimize reconstruction error

Example: MNIST

Hinton-Salakhutdinov (2006):

- First row: original images
- Second row: reconstruction using stacked autoencoder with layers of size 784-1000-500-250-30
- Third and fourth rows: reconstruction using two variants of PCA, each with 30 components

Prediction with expert advice

CSE 250B

Theory of generalization

Recall the usual statistical framework for understanding machine learning.

All data come from an underlying, albeit unknown, distribution P.

- Input space $\mathcal{X}$
- Label space $\mathcal{Y}$
- P is a distribution on $\mathcal{X} \times \mathcal{Y}$

Why does learning work?

- Training data, if there is enough of it, is representative of P.
- A classifier that does well on the training data is likely to do well on P, and thus on future test instances drawn from P.

Rich mathematical theory of how much training data is enough, for different types of classifiers.

The statistical model does not always hold

1. There is an underlying distribution P but instances aren't independent.
   Can you think of such cases?
2. Training distribution $\neq$ test distribution.

   Two well-studied settings:
   - Distributional drift: P is gradually changing.
     People's interests change, language evolves, fashions come and go.
   - Domain adaptation: test distribution known to be different from training distribution.
     E.g. A speech recognizer trained on pristine studio recordings is used in a noisy environment.
3. Adversarial settings in which a distributional assumption is questionable.
   E.g. Spam.

Is learning possible in the absence of any distribution assumption?

A more adversarial model

No distributional assumption. Data can be arbitrary, even adversarial.

An online learning process:

- For $t = 1, 2, \ldots$:
  - See next data point $x_t \in \mathcal{X}$
  - Make a prediction of its label: $\widehat{y}_t$
  - See the actual label $y_t \in \mathcal{Y}$
  - Incur loss $\ell(y_t, \widehat{y}_t)$

The learner is also given a set $\mathcal{H}$ of classifiers $\mathcal{X} \to \mathcal{Y}$.
Goal: do (almost) as well as the best classifier from $\mathcal{H}$ *in hindsight*.

Regret of the learner at any time T:

$$\text{regret}(T) = \underbrace{\sum_{t=1}^{T} \ell(y_t, \widehat{y}_t)}_{\text{loss of learner}} - \min_{h \in \mathcal{H}} \underbrace{\sum_{t=1}^{T} \ell(y_t, h(x_t))}_{\text{loss of } h}.$$

What would the regret need to be for us to be convinced that "learning" has taken place?

An illustrative bad case

Binary prediction ($\mathcal{Y} = \{0, 1\}$) under 0-1 loss.

- For $t = 1, 2, \ldots$:
  - See next data point $x_t \in \mathcal{X}$
  - Make a prediction of its label: $\widehat{y}_t \in \{0, 1\}$
  - See the actual label $y_t \in \{0, 1\}$
  - Incur loss $1(y_t \neq \widehat{y}_t)$

Suppose $\mathcal{H}$ has just two hypotheses:

h_0 always predicts 0

h_1 always predicts 1

The adversary generating the data can always pick $y_t \neq \widehat{y}_t$. At time T:

- Learner has loss T
- Best hypothesis in $\mathcal{H}$ has loss at most $T/2$

Therefore regret $\geq T/2$.

To get regret $o(T)$, use **randomization** to protect against the adversary.

Randomized prediction

Suppose $|\mathcal{H}| = N$. Think of these hypotheses as N "experts".
Want to do (nearly) as well as the best expert in hindsight.

- For $t = 1, 2, \ldots$:
  - See next data point $x_t \in \mathcal{X}$
  - Get predictions of the experts on x_t
  - Choose a distribution over the expert predictions:

$$p_t = (p_t^{(1)}, \ldots, p_t^{(N)})$$

  - See the actual label $y_t \in \mathcal{Y}$
  - Incur *expected* loss $\mathbb{E}_{i \sim p_t}[\ell_t^{(i)}]$, where $\ell_t^{(i)} = \ell(y_t, h_i(x_t))$ is the loss incurred by the ith expert

Expected regret of the learner at time T:

$$\text{regret}(T) = \underbrace{\sum_{t=1}^{T} \sum_{i=1}^{N} p_t^{(i)} \ell_t^{(i)}}_{\text{expected loss of learner}} - \underbrace{\min_{1 \leq i \leq N} \sum_{t=1}^{T} \ell_t^{(i)}}_{\text{loss of best expert}}.$$

The Hedge algorithm: example

Hedge:

- Start by weighing all experts equally.
- Any time an expert makes a mistake, reduce its weight by a *multiplicative* factor.

Expert	Wt	Loss	Wt	Loss	Wt	Loss	Wt	Loss
1	$\frac{1}{4}$	0	$\frac{1}{3}$	0	$\frac{4}{9}$	0	$\frac{8}{15}$	1
2	$\frac{1}{4}$	0	$\frac{1}{3}$	1	$\frac{2}{9}$	0	$\frac{4}{15}$	1
3	$\frac{1}{4}$	1	$\frac{1}{6}$	1	$\frac{1}{9}$	1	$\frac{1}{15}$	1
4	$\frac{1}{4}$	1	$\frac{1}{6}$	0	$\frac{2}{9}$	1	$\frac{2}{15}$	1
Hedge		$\frac{1}{2}$		$\frac{1}{2}$		$\frac{1}{3}$		1

The Hedge algorithm

Definitions at time t:

- Loss of ith expert is $\ell_t^{(i)} \in [0, 1]$
- Cumulative loss of ith expert is $L_t^{(i)} = \sum_{t' \leq t} \ell_{t'}^{(i)}$

The Hedge(η) algorithm:

- For $t = 1, 2, \ldots$:
  - Choose a distribution p_t over the N expert predictions:

$$p_t^{(i)} \propto e^{-\eta L_{t-1}^{(i)}} \quad \text{(that is, } p_t^{(i)} \propto p_{t-1}^{(i)} e^{-\eta \ell_{t-1}^{(i)}}\text{)}$$

  - Incur expected loss $\mathbb{E}_{i \sim p_t}[\ell_t^{(i)}]$

Theorem. Let L_T^{Hedge} denote the cumulative expected loss of this algorithm upto time T. For any T,

$$L_T^{\text{Hedge}} \leq \min_{1 \leq i \leq N} \frac{\eta L_T^{(i)} + \ln N}{1 - e^{-\eta}}.$$

A simple case

Let's say the labels and predictions are binary, with 0-1 loss.
Choose $\eta = \ln 2$ so that $e^{-\eta} = 1/2$.

The Hedge(η) algorithm:

- For $t = 1, 2, \ldots$:
  - Choose a distribution p_t over the N expert predictions:

$$p_t^{(i)} \propto \left(\frac{1}{2}\right)^{\#(\text{mistakes made by expert } i \text{ so far})}$$

  - Loss incurred = total weight $p_t^{(\cdot)}$ of experts incorrect on this round

Theorem. Let L_T^{Hedge} denote the cumulative expected loss of this algorithm upto time T. For any T,

$$L_T^{\text{Hedge}} \;\leq\; (2\ln 2) \min_{1 \leq i \leq N} L_T^{(i)} \;+\; 2\ln N.$$

Is there a better choice of η?

Tuning η

Suppose $\min_i L_i^T \leq \widetilde{L}$ (certainly this is at most T).

Setting

$$\eta = \ln\left(1 + \sqrt{\frac{2\ln N}{\widetilde{L}}}\right)$$

gives

$$\begin{aligned} L_{\text{Hedge}}^T &\leq \min_i L_i^T + \sqrt{2\widetilde{L}\ln N} + \ln N \\ &\leq \min_i L_i^T + \sqrt{2T\ln N} + \ln N. \end{aligned}$$

The regret is $O(\sqrt{T\ln N})$ . . . can we do better?

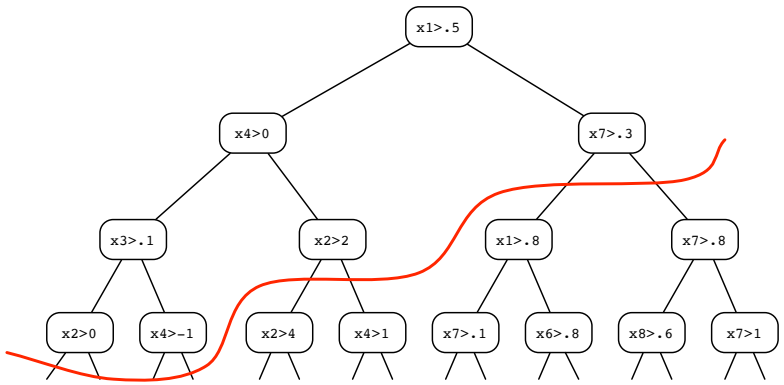
A lower bound on the regret

Consider a simple adversary: on any round, he selects a label 0 or 1 by flipping a fair coin.

1. What is the expected loss of the learner on each round, regardless of strategy?
   Exactly $1/2$. Therefore, expected loss after T steps is $T/2$.
2. Now suppose the hypothesis class $\mathcal{H}$ contains the always-zero hypothesis h_0 and the always-one hypothesis h_1. What is the cumulative loss of the best classifier in hindsight?
   By random fluctuation, at time T, the number of 1's seen will differ from $T/2$ by about $\sqrt{T}$. Therefore, either h_0 or h_1 has cumulative loss about $T/2 - \sqrt{T}$.

In short: a regret of $\sqrt{T}$ is inevitable.
Can extend the argument to make this lower bound $\sqrt{T\ln N}$.

Application: pruning a decision tree

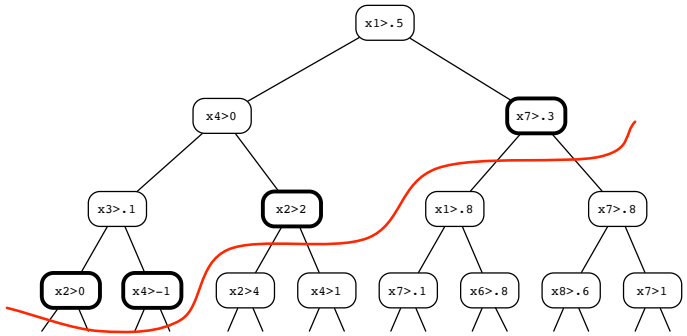


Idea: have an expert for each possible pruning. Use Hedge to always predict (nearly) as well as the best pruning!

Problem: there are exponentially many prunings...

Pruning a decision tree: computational efficiency

The weight of a pruning depends only on the error statistics at its leaf nodes.



No need to keep a separate weight for each pruning:

- Keep error count at each node.
- Obtain weights of prunings and make predictions by combining these efficiently, using dynamic programming.

More online learning

1. Bandit problems
2. Contextual bandits
3. Online linear classification
4. Online convex optimization
5. ... and a lot more

Semisupervised and active learning

CSE 250B

Supervised learning of classifiers

Example: build a system that identifies which credit card transactions are fraudulent.

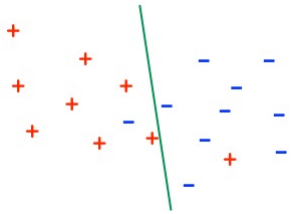
The input space: $\mathcal{X} = \{\text{descriptions of credit card transactions}\}$

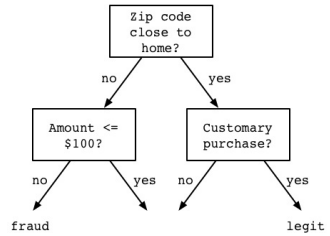
The label space: $\mathcal{Y} = \{\text{legitimate, fraudulent}\}$

The training set: A bunch of labeled examples (x, y).

Supervised learning: Use training data to learn a prediction rule (or classifier) $f : \mathcal{X} \to \mathcal{Y}$. We will use this rule on future test cases.

Many types of classifiers:





A statistical framework for learning

An **unknown underlying distribution** $\mathbb{P}$ on the (data, label) space.
E.g. Distribution of nationwide credit card transactions.
The training data are random draws from this distribution, as are future test cases.

Hypothesis class H of candidate classifiers.
Target: the $h^* \in H$ that has the smallest probability of error on $\mathbb{P}$.

Get n samples from $\mathbb{P}$, choose $h_n \in H$ that does well on these.
E.g. Pick the h_n that makes the fewest mistakes on the training data.

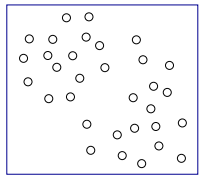
Consistency: as n grows, $h_n \to h^*$.
Sample complexity: Roughly how many samples are needed to get a reasonably good estimate?

Exploiting unlabeled data

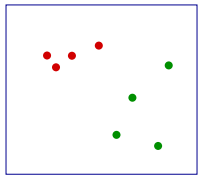
A lot of unlabeled data is plentiful and cheap, eg.

- documents off the web
- speech samples
- images and video

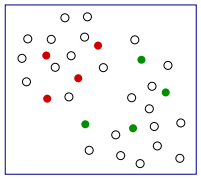
But labeling can be expensive.



Unlabeled points



Supervised learning



Semisupervised and active learning

Semisupervised and active learning

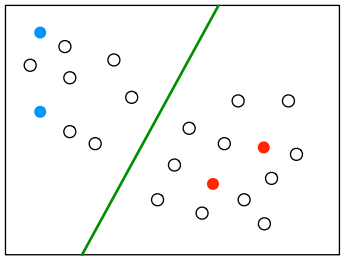
1. Three ideas for semisupervised learning
2. Active learning
   - Using importance weighting
   - Exploiting cluster structure

Semisupervised learning 1

We are given:

- A large collection of unlabeled points
- Labels for some of them

Goal: learn a good classifier.

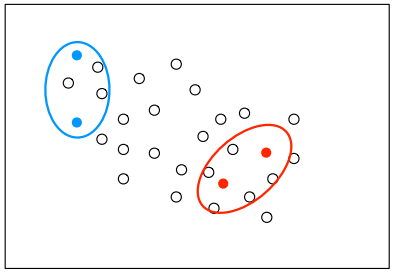


Idea: find a **linear separator** that fits the labels and has a large margin on the unlabeled points.

Problem: realizing this scheme tends to be computationally intractable.

Semisupervised learning 2

Idea: Given unlabeled data U, labeled data L, use a **generative model** to infer the labels of some points in U.



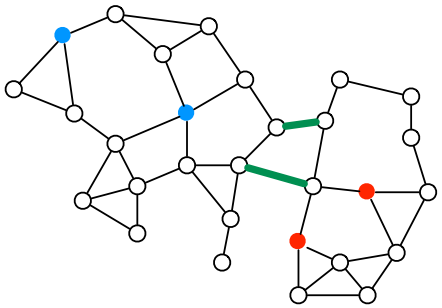
Repeat until L no longer changes:

- Fit a generative model to L
- For each $x \in U$:
  - The generative model assigns x a distribution $p(\cdot|x)$ over labels
  - If $p(y|x) > 1 - \epsilon$: add (x, y) to L, remove x from U

Does this fall into self-justifying local optima?

Semisupervised learning 3

Idea: Propagate labels on a **neighborhood graph** built on $U \cup L$.

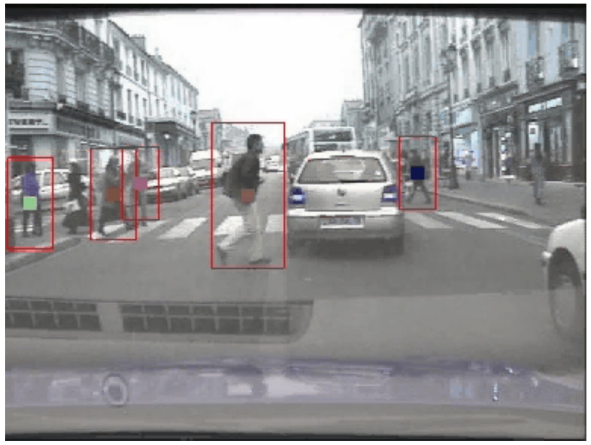


For instance, divide the nodes into two groups so that:

- The $+$ labels are in one group and the $-$ labels in the other
- The number of edges between the two groups is minimized

How many labels are needed for a non-parametric scheme like this to work?

Active learning example: pedestrian detection



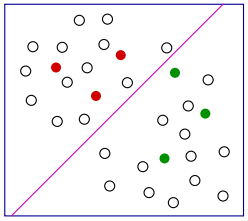
Typical heuristics for active learning

Start with a pool of unlabeled data

Pick a few points at random and get their labels

Repeat

- Fit a classifier to the labels seen so far
- Query the unlabeled point that is closest to the boundary
- (or most uncertain, or most likely to decrease overall uncertainty,...)



How to analyze such schemes?

Recall: statistical framework for learning

Unknown, underlying distribution $\mathbb{P}$ on the (data, label) space.
Hypothesis class H of candidate classifiers.
Target: the $h^* \in H$ that has fewest errors on $\mathbb{P}$.

Get n samples from $\mathbb{P}$, choose $h_n \in H$ that does well on these.

We'd like: $h_n \to h^*$, as rapidly as possible.

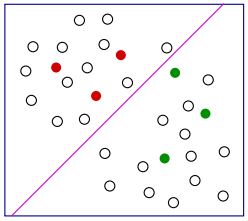
Typical heuristics for active learning

Start with a pool of unlabeled data

Pick a few points at random and get their labels

Repeat

- Fit a classifier to the labels seen so far
- Query the unlabeled point that is closest to the boundary
- (or most uncertain, or most likely to decrease overall uncertainty,...)



Biased sampling: the labeled points are not representative of the underlying distribution.

Sampling bias

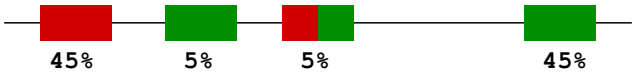
Start with a pool of unlabeled data

Pick a few points at random and get their labels

Repeat

Fit a classifier to the labels seen so far
Query the unlabeled point that is closest to the boundary
(or most uncertain, or most likely to decrease overall uncertainty,...)

Example: data in $\mathbb{R}$, $H = \{\text{thresholds}\}$.



Even with infinitely many labels, converges to a classifier with 5% error instead of the best achievable, 2.5%. *Not consistent.*

Manifestation in practice: overlooked clusters in high dimension.

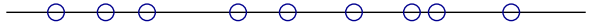
Can adaptive querying really help?

Threshold functions on the real line:

$H = \{h_w : w \in \mathbb{R}\}$
$h_w(x) = 1(x \geq w)$

Supervised: for misclassification error $\leq \epsilon$, need $\approx 1/\epsilon$ labeled points.

Active learning: instead, start with $1/\epsilon$ *unlabeled* points.



Binary search: need just $\log 1/\epsilon$ labels, from which the rest can be inferred. *Exponential improvement in label complexity.*

Challenges: Nonseparable data? Other hypothesis classes?

Two approaches to active learning

1. Use importance weighting.
2. Exploit cluster structure.

Classification with loss function

Given training data $(x_1, y_1), \ldots, (x_n, y_n)$, we typically choose the classifier $h \in H$ that makes the fewest errors:

$$\text{errors}(h) = \sum_{i=1}^{n} 1(h(x_i) \neq y_i)$$

Or that minimizes some other *loss function*:

$$L(h) = \sum_{i=1}^{n} \ell(h(x_i), y_i),$$

Common loss functions:

- $0-1$ loss: $\ell(y, y') = 1(y \neq y')$
- Logistic loss: $\ell(y, y') = \ln(1 + e^{-yy'})$
- Squared loss: $\ell(y, y') = (y - y')^2$
- Hinge loss: $\ell(y, y') = (1 - yy')_+$

Importance weighted active learning

Active learning boilerplate:

1. Initialize $S = \emptyset$.
2. For $t = 1, 2, \ldots, T$:
   - Receive a new point x_t.
   - Choose a probability p_t.
   - With probability p_t: query label y_t and add $(x_t, y_t, 1/p_t)$ to S.
3. Return classifier $h \in H$ minimizing the weighted empirical loss

$$L_T(h) = \sum_{(x,y,w) \in S} w\, \ell(h(x), y).$$

Consistency of importance-weighted active learning

1. Initialize $S = \emptyset$.
2. For $t = 1, 2, \ldots, T$:
   - Receive a new point x_t.
   - Choose a probability p_t.
   - With probability p_t: query label y_t and add $(x_t, y_t, 1/p_t)$ to S.
3. Return the classifier $h \in H$ minimizing the weighted empirical loss $L_T(h) = \sum_{(x,y,w) \in S} w\, \ell(h(x), y)$.

Define $Q_t = 1(y_t \text{ is queried})$. Then $\mathbb{E}[Q_t | p_t] = p_t$, and

$$\mathbb{E}[L_T(h)] \;=\; \mathbb{E}\left[\sum_{t=1}^{T} \frac{Q_t}{p_t} \ell(h(x_t), y_t)\right] \;=\; \mathbb{E}_{(X,Y)\sim\mathbb{P}} \ell(h(X), Y) \;=\; L(h).$$

If H is finite and all $p_t \geq p_{min}$, then with probability at least $1 - \delta$,

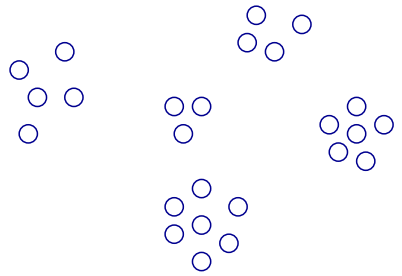
$$\max_{h \in H} |L_T(h) - L(h)| \;\leq\; \sqrt{\frac{2 \ln |H|/\delta}{T p_{min}}}.$$

Two approaches to active learning

1. Use importance weighting.
2. Exploit cluster structure.

Exploiting cluster structure in data

Suppose the unlabeled data looks like this.



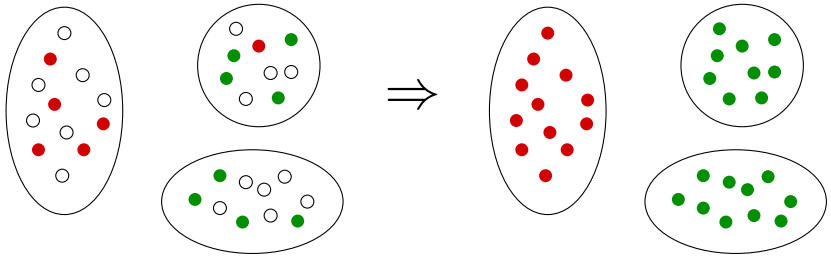
Then perhaps we just need five labels.

Challenges: In general, the cluster structure (i) is not so clearly defined and (ii) exists at many levels of granularity. And (iii) the clusters may not be pure in their labels.

Exploiting cluster structure in data

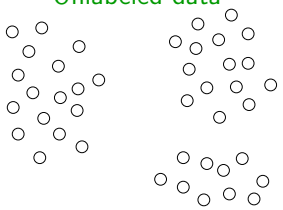
Basic primitive:

- Find a clustering of the data
- Sample a few *randomly-chosen* points in each cluster
- Assign each cluster its majority label
- Now use this fully labeled data set to build a classifier

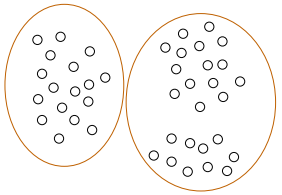


Finding the right granularity

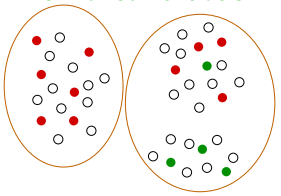
Unlabeled data



Find a clustering



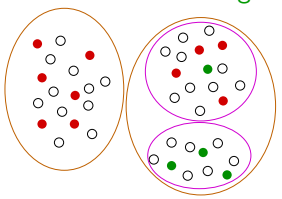
Ask for some labels



(random sampling within clusters)

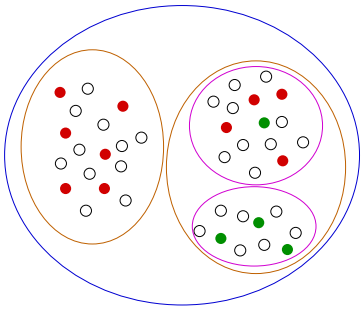
Now what?

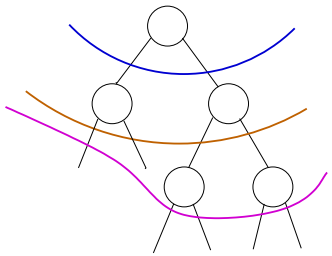
Refine the clustering



Queried points are also randomly distributed within the new clusters.

Using a hierarchical clustering

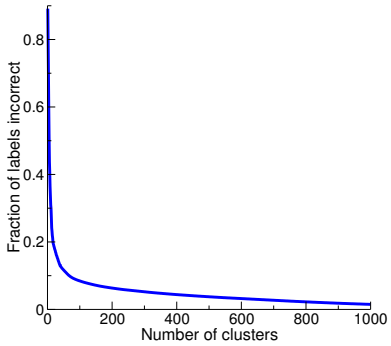


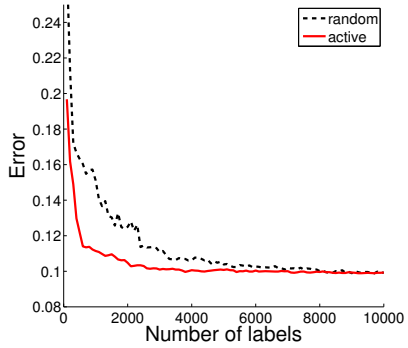


- Always work with some pruning of the hierarchy: a clustering induced by the tree. Pick a cluster (intelligently) and query a *random* point in it.
- For each tree node (i.e. cluster) v maintain: (i) majority label $L(v)$; (ii) empirical label frequencies $\widehat{p}_{v,l}$; and (iii) confidence interval on label frequencies, $[p^{lb}_{v,l}, p^{ub}_{v,l}]$

Example: MNIST digits

Hierarchy built using Ward's agglomerative clustering (k-means cost function) with Euclidean distance.





Open problem: interactively finding a good hierarchical clustering.