Machine Learning

CS 165B

Prof. Matthew Turk

Monday, April 4, 2016

Today

- The ingredients of machine learning (Ch. 1)

Notes

- HW#1 posted, due on Friday at 4:30pm
  - No programming problems
  - Turn in via (1) homework box in HFH or (2) GauchoSpace
  - If turned in via GauchoSpace, must be typeset – NO pictures or scans!
- Grading
  - 5 homeworks: lowest score worth 10%, others worth 22.5% each
  - Overall grading is curved somewhat – don't assume X% = Y grade
- Course registration
  - ~12 new registrations will be added today
  - Waitlist is at 25 (will be 13)

Notes

CS COLLOQUIUM

Ce Zhang, Stanford

Wednesday, April 6

3:30pm

CS conf. room, 1132 HFH

"DeepDive: A Data Management System for Machine Learning Workloads"

CS COLLOQUIUM

Sameer Singh, Washington

Monday, April 11

3:30pm

CS conf. room, 1132 HFH

"Interactive Machine Learning for Information Extraction"

Notes

CS DISTINGUISHED LECTURE

Michael Jordan, Berkeley

Friday, April 8

11:00am

Corwin Pavilion

“On Computational Thinking, Inferential Thinking, and Data Science”

Tasks: predictive and descriptive

- The most common ML tasks are predictive, aiming to predict/estimate a target variable from features:
  - Binary and multi-class classification: categorical target
    - Learn decision boundaries
  - Regression: numerical target
    - Learn relationship (a real-valued function) between input and output spaces

- Descriptive tasks are concerned with exploiting underlying structure in the data, finding patterns:
  - No specific problem to solve per data element
  - Goal: discover “interesting things” about the data
  - E.g., (descriptive) clustering
    - Grouping data without prior information

Models

- Machine learning models can be distinguished according to their main intuition:
  - Geometric models use intuitions from geometry such as separating (hyper-)planes, linear transformations and distance metrics
  - Probabilistic models view learning as a process of reducing uncertainty, modelled by means of probability distributions
  - Logical models are defined in terms of easily interpretable logical expressions

- Alternatively, they can be characterized by their *modus operandi* (i.e., the model style):
  - Grouping models divide the instance space into segments (at training time); in each segment a very simple (e.g., constant) model is learned
  - Grading models learning a single, global model over the instance space

Grouping and grading models

- Distinction: how they handle the instance space
- Grouping models break up the instance space into groups or segments
  - Don’t distinguish between individual instances within each segment
  - Thus, a finite (possibly coarse) resolution of the instance space
  - Within a segment, assign the same output class to all instances – e.g., based on a majority vote
  - Key issue: determining good segment boundaries
- Grading models do not segment the instance space – they form a single global model (function) over the complete instance space
  - Infinite resolution (in theory) possible; can distinguish between arbitrary instances

Grouping and grading models (cont.)

For example, consider course grades:

- A machine learning program may predict the grade for CS165B based on the grades for CS165A and PSTAT120A

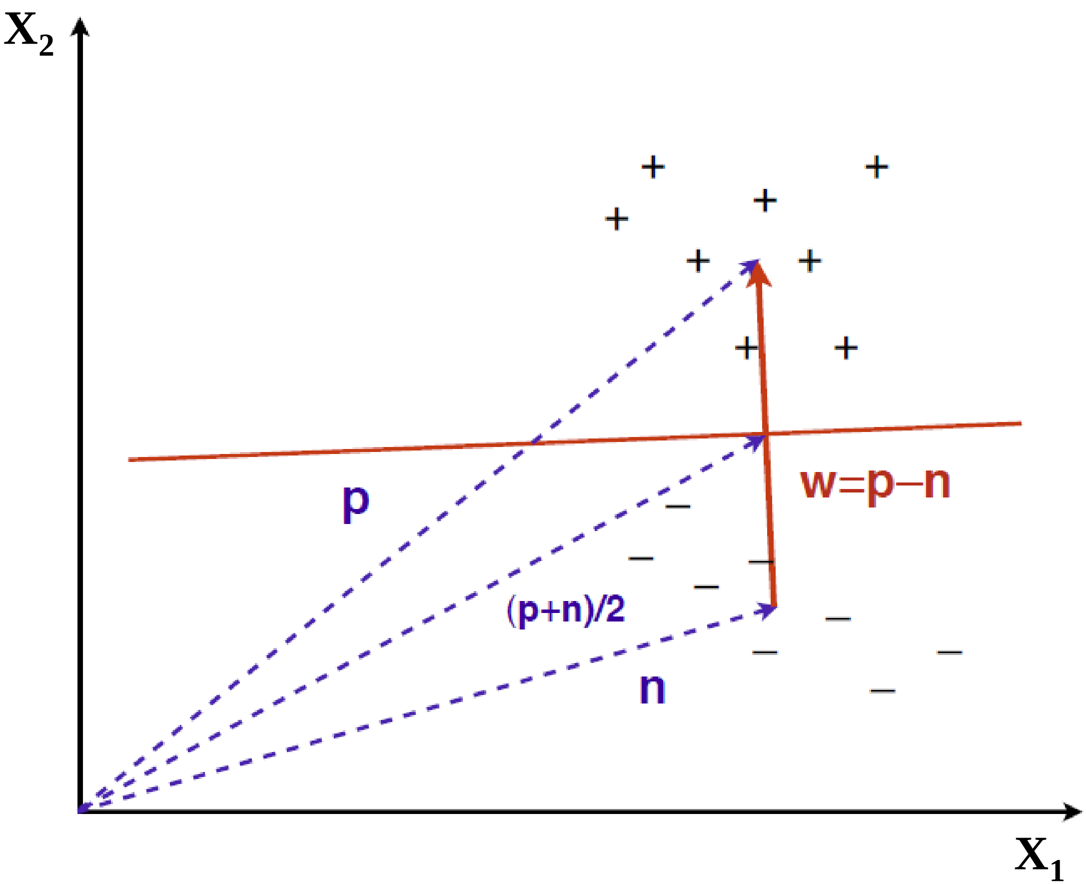
- Grouping model:
  - Inputs are letter grades, A-F
- Grading model:
  - Inputs are real-valued numeric scores, $0 \leq x \leq 100$

This distinction is an observation, something to consider when designing a ML system – not a specific method

Many systems are somewhere in between (combine the two)

Basic linear classifier

Constructs a linear decision boundary halfway between the positive and negative centers of mass of the two classes

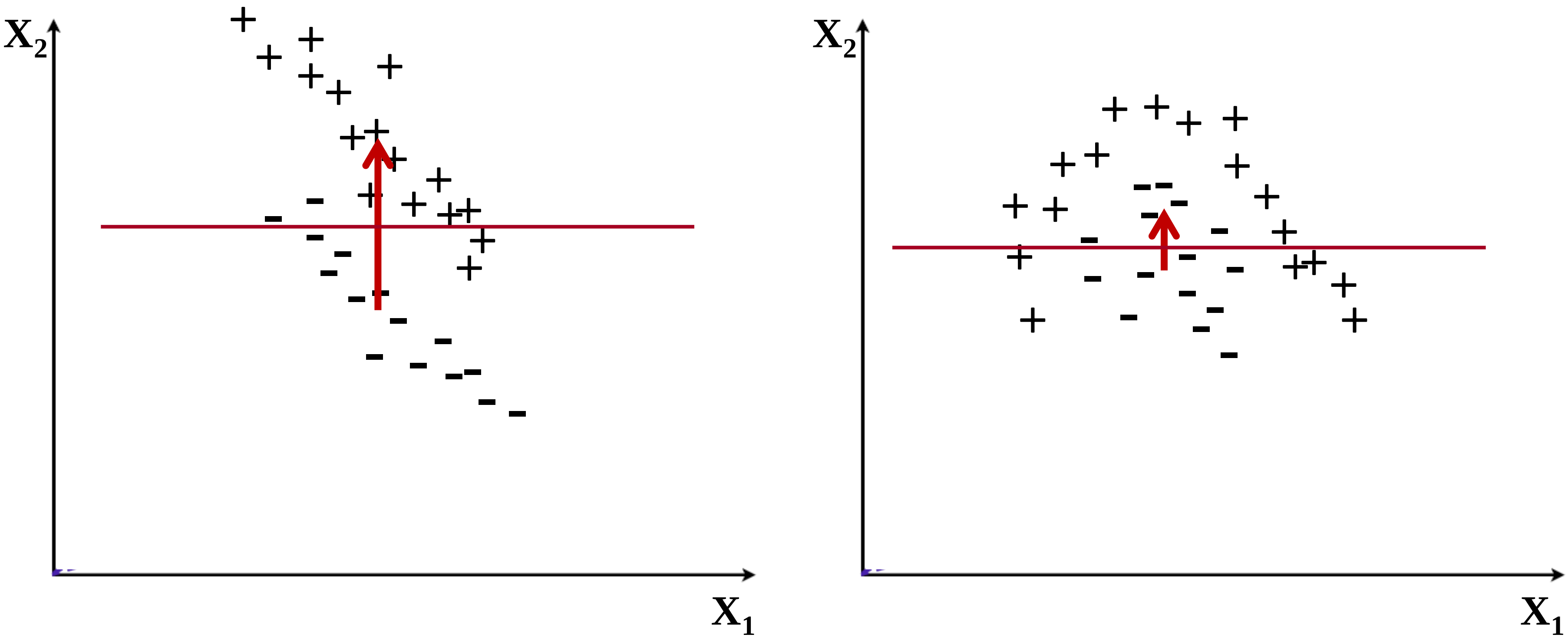


$$f(x) = \begin{cases} 1 & \text{if } \mathbf{x} \cdot \mathbf{w} > t \\ 0 & \text{otherwise} \end{cases}$$

How to compute t ?

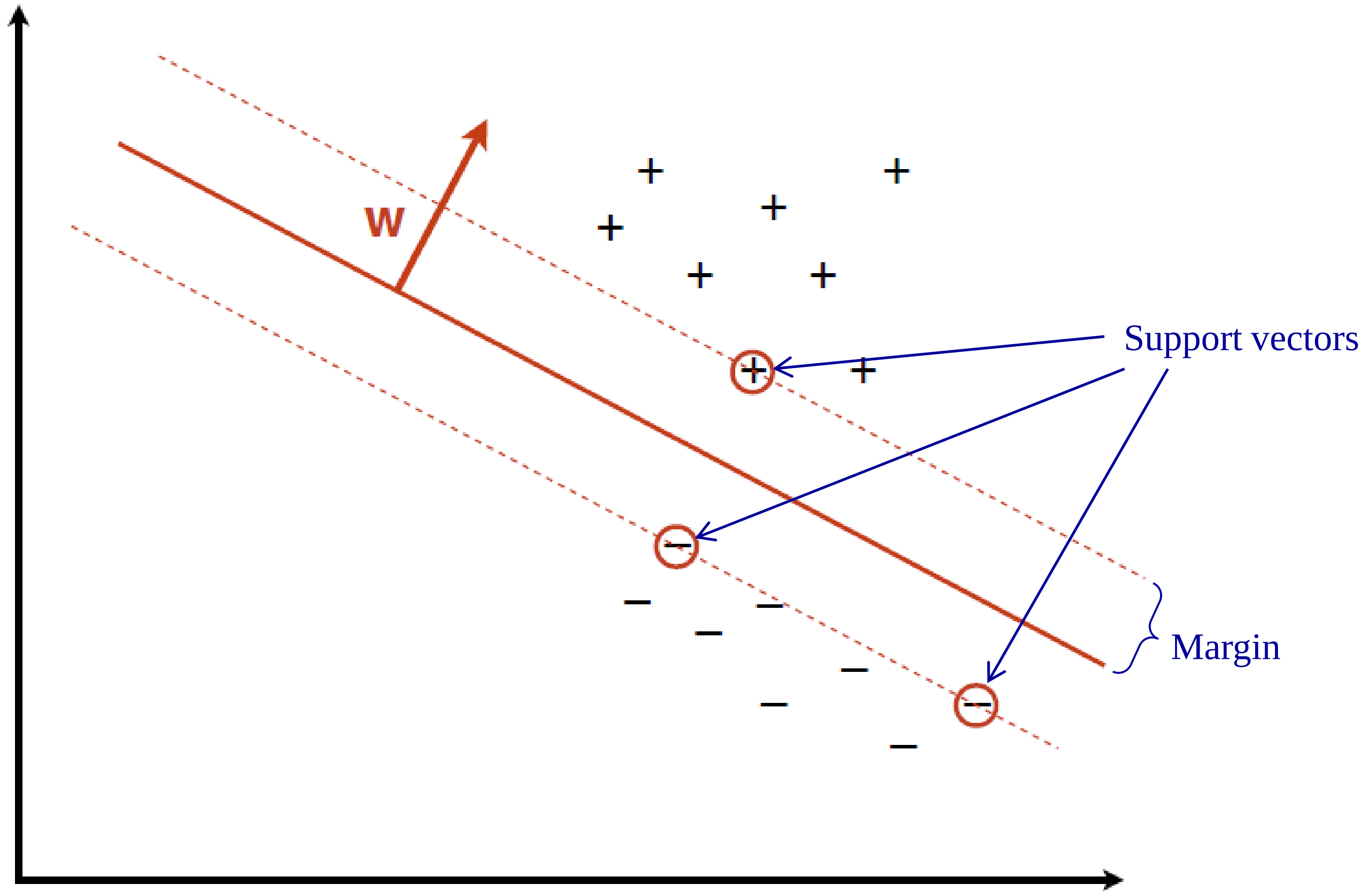
Basic linear classifier (cont.)

That strategy wouldn't work so well in these situations:



Support Vector Machine (SVM) classifier

SVM learns the optimal decision boundary from linearly separable data, maximizing the ***margin***



Probabilistic models

- In general, probabilistic models aim to model the relationship between the feature values **X** and the target variables **Y** using probability distributions
- Predict **Y** based on **X** and the *posterior distribution* **P(Y | X)**
- Using Bayes' Rule

Likelihood

Prior

Posterior

$$\underbrace{P(Y|X)}_{\text{Posterior}} = \frac{\overbrace{P(X|Y)}^{\text{Likelihood}}\,\overbrace{P(Y)}^{\text{Prior}}}{P(X)}$$

- Decision rule: Choose **Y** that maximizes the value of **P(Y | X)**
  - Known as the *maximum a posteriori (MAP)* rule, or *MAP estimation*
- Decision rule: Choose **Y** that maximizes the value of **P(X | Y)**
  - Known as the *maximum likelihood (ML)* rule, or *maximum likelihood estimation*

Probabilistic models (cont.)

Binary classification example: I wake up in the morning and want to know whether or not it rained outside. I can look out the window and see if the grass is wet.

- Target variable (Y) – Did it rain? (binary classification task)
- Data (aka observation) (X) – Is the grass wet? (binary input variable)
- Learned models: P(X | Y) and P(Y) (if available), from prior experience

ML approach: compute the likelihood ratio

$$LR(X) = \frac{P(X|Y = rain)}{P(X|Y = \overline{rain})}$$

$$\hat{Y} = \begin{cases} 1 & if\ LR(X) > 1 \\ 0 & otherwise \end{cases}$$

MAP approach: compute the posterior odds

$$PO(X) = \frac{P(X|Y = rain)\boxed{P(Y = rain)}}{P(X|Y = \overline{rain})\boxed{P(Y = \overline{rain})}}$$

$$\hat{Y} = \begin{cases} 1 & if\ PO(X) > 1 \\ 0 & otherwise \end{cases}$$

Probabilistic models (cont.)

- The likelihood function **P(X | Y)** plays an important role in statistical machine learning
  - **P(Data | Hypotheses)**
  - Think of the likelihood function as diagnostic information
    - What are the likely symptoms of various diseases?
    - What are the likely features of a face?
    - What are the likely outcomes of various events?
- A full likelihood function is a generative model – a probabilistic model from which we can sample values of all the data variables
  - E.g., we can use **P(symptoms | diseases)** to generate samples of symptoms, given a certain disease
  - Alternative: discriminative models

Probabilistic models (cont.)

Textbook example: Spam filtering (binary classification task)

Hypotheses: spam or ham

Data: presence of certain words in the email

Viagra	lottery	$P(Y = \text{spam} \mid \text{Viagra}, \text{lottery})$	$P(Y = \text{ham} \mid \text{Viagra}, \text{lottery})$
0	0	0.31	**0.69**
0	1	**0.65**	0.35
1	0	**0.80**	0.20
1	1	0.40	**0.60**

Decision rule: Spam or ham, based on the presence of these two words

MAP, ML, …

Aside: Basic PSTAT background assumed

- You should know basic probability and statistics, including:
  - Axioms of probability
  - Events, independence, conditional independence
  - Probability distribution functions
    - Probability mass/density functions
    - Cumulative distribution function
  - Joint probability distributions
  - Conditional probability distributions
  - Marginalization
  - Bayes' Rule
  - Mean, standard deviation, variance, covariance
  - Normal/Gaussian distribution
  - Central limit theorem

Q: How many entries are there in the joint probability distribution over all the variables in the "spam or ham" problem?

Q: How many *independent* entries are in the joint probability distribution table for **P(Y | Viagra, lottery)**?

Aside: Basic Linear Algebra background assumed

- You should know basic linear algebra, including:
  - Matrix properties
    - Identity, diagonal, transpose, inverse, rank, …
  - Matrix/matrix and matrix/vector products
  - Dot products, cross product, orthogonality
  - Vector and matrix norms
  - Eigenvectors and eigenvalues
  - Singular value decomposition

Q: What matrices can be inverted?

Q: If **M** is an orthonormal matrix, what is $\mathbf{M}^T\mathbf{M}$?

Probability tables

- How do we get the values in the probability tables?
- In many cases, we collect data and estimate the values directly from the data
  - I.e., counting

P(Viagra=0, lottery=1 | Y=spam)

In the database, of all the spam emails, what percentage contain the word “Viagra” but not the word “lottery”?

P(Viagra=0, lottery=1 | Y=ham)

In the database, of all the non-spam emails, what percentage contain the word “Viagra” but not the word “lottery”?

Q: What do these two probabilities sum to?

A: I have no idea! (Probably not 1)

Logical models

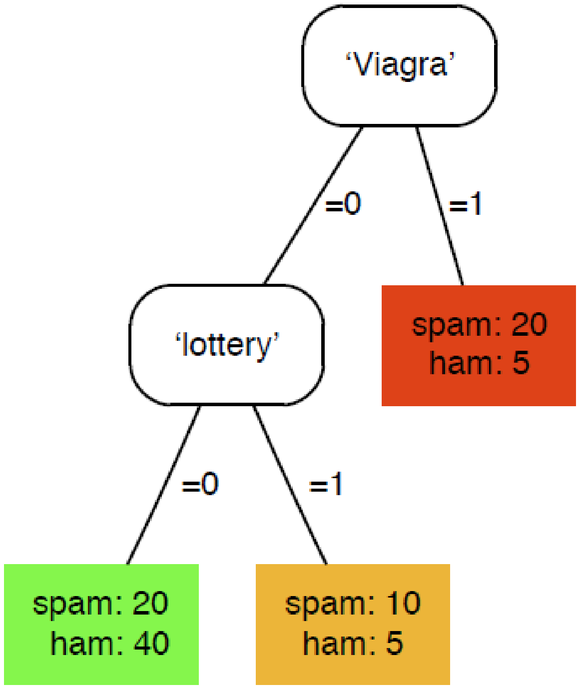
- Geometric and probabilistic models don't necessarily translate to human-understandable rules
- Logical models focus at the level of human reasoning
  – Often provide explanations for their results
- Classical AI models encapsulate logical rules and relationships for deductive reasoning, e.g.:
  – Propositional logic
    - Simple declarative propositions
    - Boolean logic, basic and derived rules
    - E.g., modus ponens: $((p \rightarrow q) \wedge p) \Rightarrow q$
  – First-order (predicate) logic
    - Adds predicates, quantification
    - Expresses much broader semantics
    - E.g., $\forall x \,\mathrm{Man}(x) \Rightarrow \mathrm{Mortal}(x)$

Logical models (cont.)

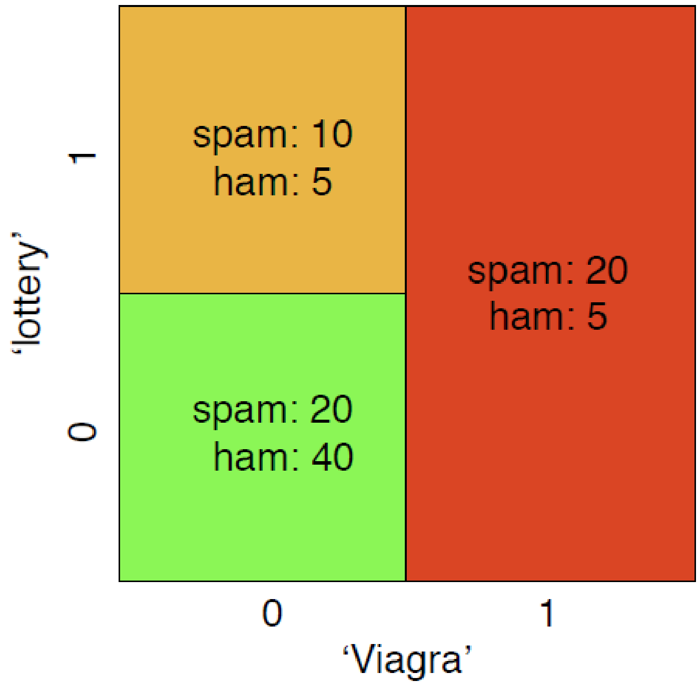
- Logical models in ML are often organized in tree structures – feature trees – that iteratively partition the space of all possible inputs (the *instance space*)
  - The nodes represent decisions based on feature values
  - The leaves correspond to regions of the instance space – i.e., groups of feature values
- Feature trees whose leaves are labelled with classes are called ***decision trees***

Feature trees

A feature tree for Spam vs. Ham



The partitioned instance space

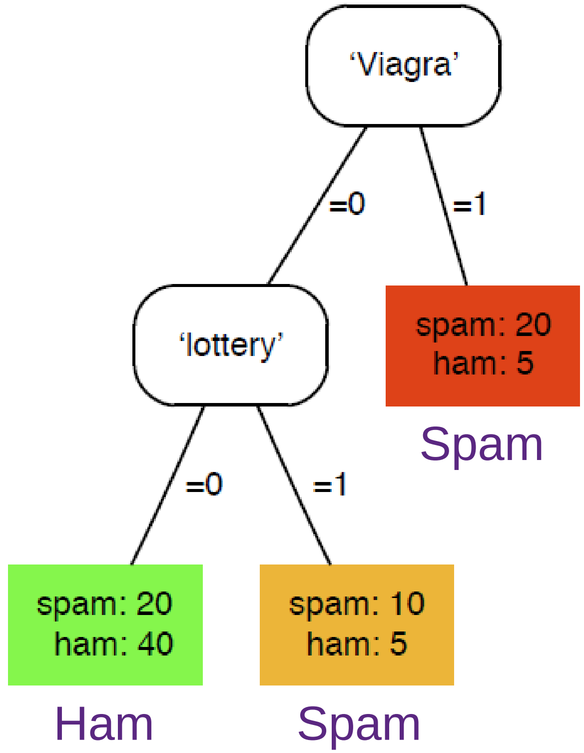


The numbers correspond to the number of emails in each bin

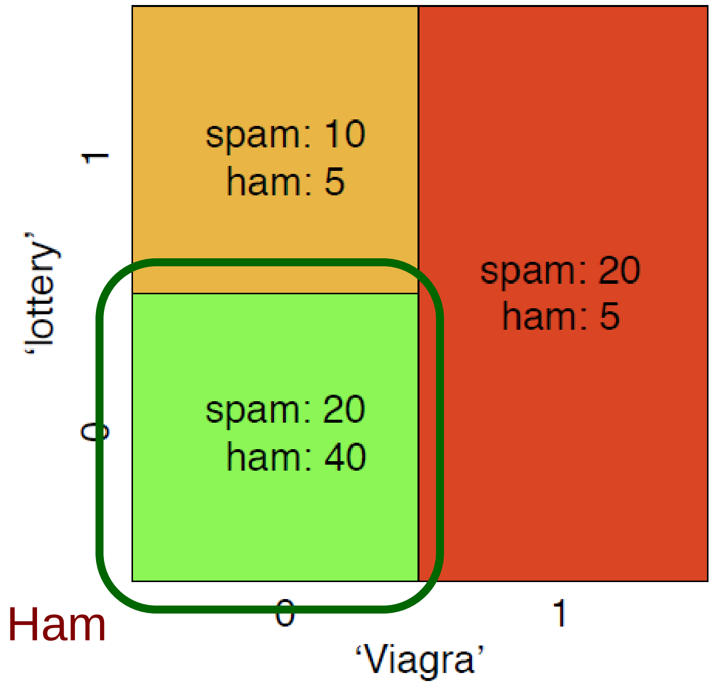
What about Viagra=1 ∧ lottery=0 ? Viagra=1 ∧ lottery=1 ?

Decision trees

A decision tree for Spam vs. Ham



The partitioned instance space



Key question for ML: How to construct a (good) decision tree from data?

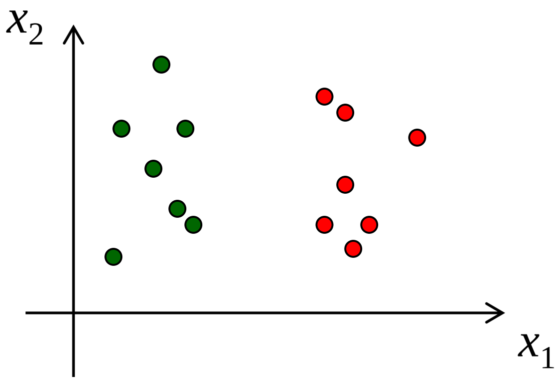
Features

- A machine learning model is only as good as its features
  - *Garbage in, garbage out!*
- Features are measurements performed on instances
  - Multiple features for an instance comprise a feature vector
  - Most often numerical, but not always
  - E.g., a feature could be "the most frequent word in the text"
- Typical feature types:
  - Boolean
  - Integers
  - Real numbers
  - Sets
- What other features might you use to classify "spam or ham"?
- For a book recommender system?

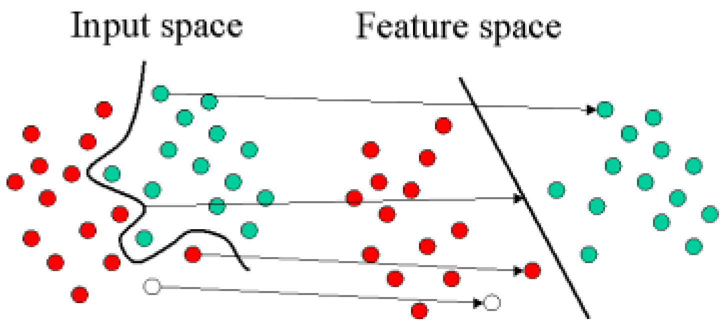
Feature construction

- In some ML problems, the features are fixed, given to you
- But in others, feature construction may be the most important part of your solution
- The "raw" features may not be best for the problem
  - They may have irrelevant dimensions
  - They may depend on irrelevant parameters
  - Some features may be particularly noisy (unreliable)
- We want features that:
  - Encapsulate the key similarities and differences in the data
  - Are robust to irrelevant parameters and transformations
  - Have a high signal-to-noise ratio
- Often, the first step in a ML problem is to transform the features into a new feature space

Feature construction/transformation



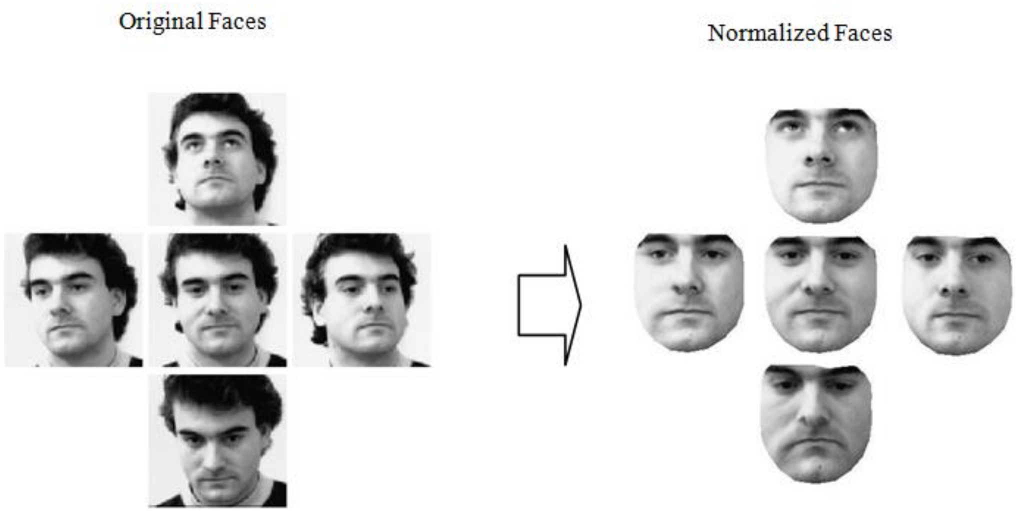
x_2 has no useful information for classifying, so transform the feature space by projecting onto the x_1 axis



Transformation (ϕ) to make the feature space linearly separable

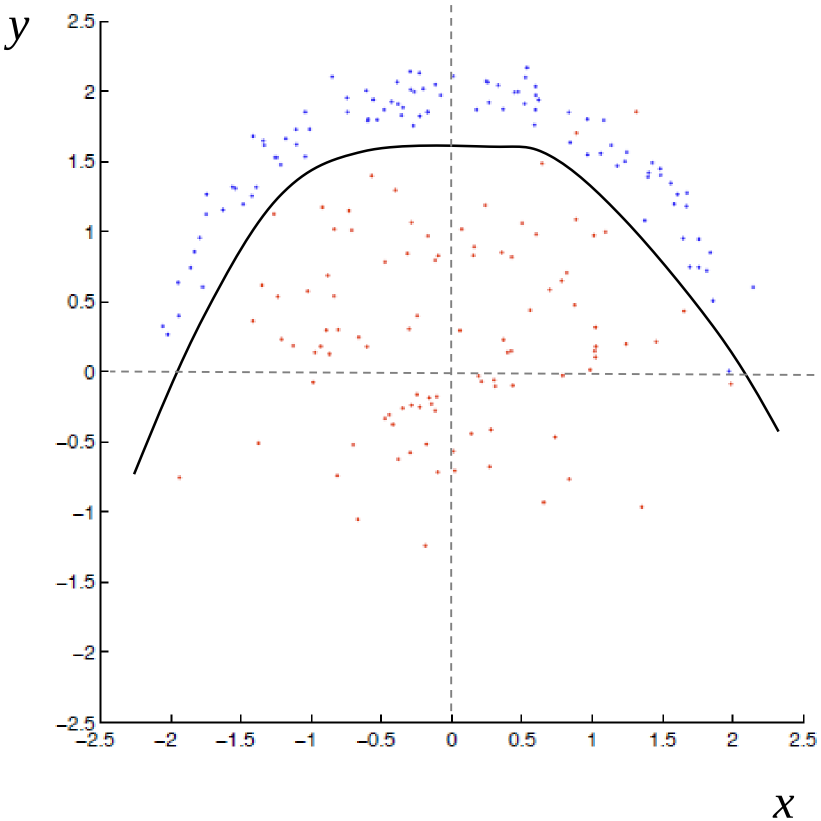
Transformation (ϕ) to make the measurements in the appropriate coordinate system

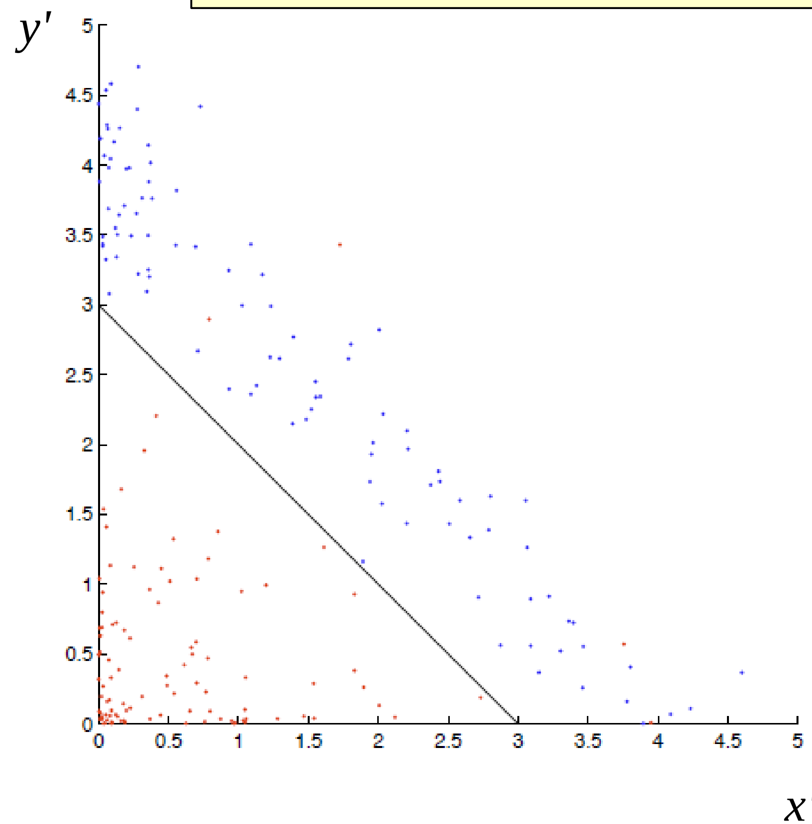
- Align the faces



Feature construction/transformation

$(x',y') = \phi(x, y) = (x^2, y^2)$





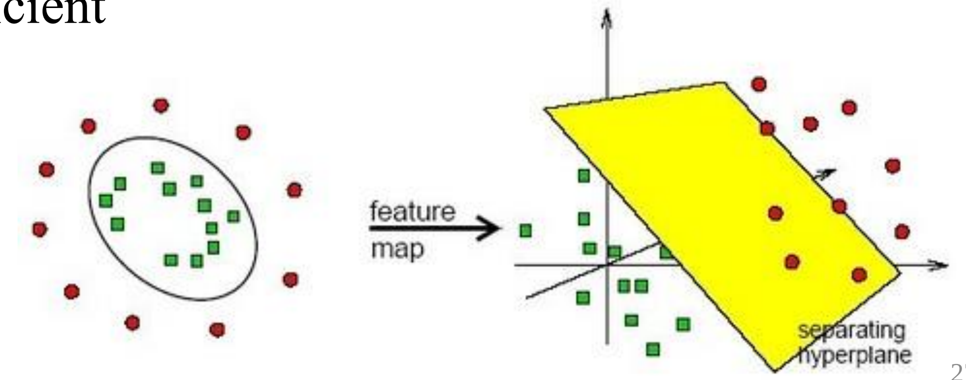
1. Transform features to new feature space – mapping ϕ from (x, y) to (x',y')
2. Perform linear classification

The kernel trick

- In machine learning, the "kernel trick" is a way of mapping features into another (often higher dimensional) space to make the data linearly separable, *without having to compute the mapping explicitly*.

- The dot product operation in a linear classifier $\boldsymbol{x_1} \cdot \boldsymbol{x_2}$ is replaced by a kernel function $\kappa(\boldsymbol{x_1}, \boldsymbol{x_2})$ that computes the dot product of the values $(\boldsymbol{x_1}', \boldsymbol{x_2}')$ in the new (linearly separable) space.
  - Again, without having to compute the mapping from $(\boldsymbol{x_1}, \boldsymbol{x_2})$ to $(\boldsymbol{x_1}', \boldsymbol{x_2}')$
  - So it's both effective and efficient

- Let's see an example….



The kernel trick

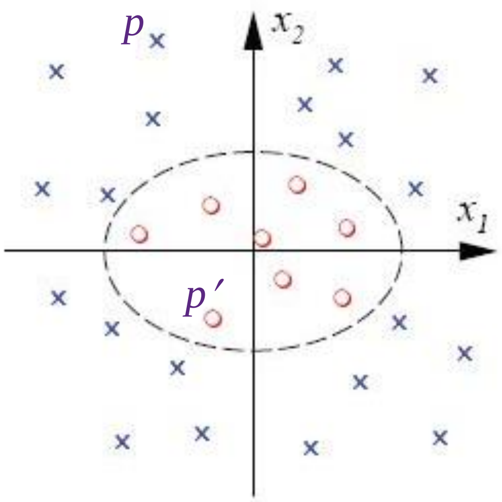
- In the original feature space, the two classes (o's and x's) are not linearly separable
- So let's map $\boldsymbol{p} = (x_1, x_2)$ to a new space $\boldsymbol{q} = (z_1, z_2, z_3)$ via the transformation:

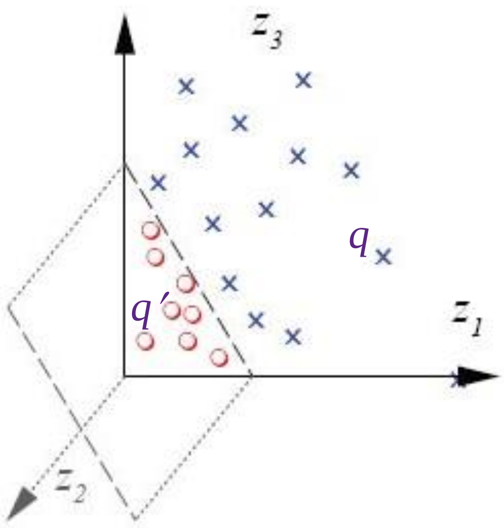
  $$z_1 = x_1^2$$
  $$z_2 = x_2^2$$
  $$z_3 = \sqrt{2}x_1x_2$$

  where, it turns out, the o's and x's are linearly separable.
- A dot product in the new space:

  $$\begin{aligned}\boldsymbol{q} \cdot \boldsymbol{q}' &= z_1 z_1' + z_2 z_2' + z_3 z_3' \\ &= x_1^2 x_1'^2 + x_2^2 x_2'^2 + \sqrt{2}x_1x_2 \sqrt{2}\, x_1' x_2' \\ &= (x_1 x_1' + x_2 x_2')^2 \\ &= (\boldsymbol{p} \cdot \boldsymbol{p}')^2 = \kappa(\boldsymbol{p_1}, \boldsymbol{p_2})\end{aligned}$$

  is merely the square of the original dot product!





Feature transformation and the kernel trick

- The kernel trick is widely used in machine learning
- Assumption: achieving linear separation is worth the effort
  - There are non-linear classifiers, but linear classification tends to be simple and fast
- Assumption: the dot product is the key computation
  - Yes, for a linear classifier
  - So we just replace the dot product with the kernel function
- How do we find the mapping that will make the data linearly separable?
  - Good question!
  - Insight into the data, trial and error, …
  - Are there principled ways to determine such a transformation?