CS 1675: Intro to Machine Learning

Intro to Classification (Nearest Neighbors, Logistic Regression, Perceptron)

Prof. Adriana Kovashka
University of Pittsburgh
September 27, 2018

Classification

- Given features ***x***, predict categorical output *y*
- For example:
  - Given attributes of a house (e.g. square footage and age built), predict whether it will be bought for the asking price or for less
  - Given temperature, predict whether it will rain, snow, or be sunny
- The rest of the course will cover different supervised approaches to classification

Plan for this lecture

- The simplest classifier: K-Nearest Neighbors
  - Algorithm and example use
  - Generalizing: Distance metrics, weighing neighbors
  - Problems: curse of dimensionality, picking K
- Logistic regression
  - Probability: review
  - Linear regression for classification?
  - Maximum likelihood solution for *logistic* regression
  - Related algorithm: perceptron

Nearest Neighbors: Key Idea

- A type of supervised learning: We want to learn to predict, for a new data point x, its label y (e.g. spam / not spam)
- Don't learn an explicit function F: X → Y
- Keep all training data {X, Y}
- For a test example x, find the training example x_i closest to it (e.g. using Euclidean distance)
- Then copy the target label y_i as the label for x

Related Methods / Synonyms

- Instance-based methods
- Exemplar methods
- Memory-based methods
- Non-parametric methods

Instance/Memory-based Learning

Four things make a memory based learner:

- *A distance metric*
- *How many nearby neighbors to look at?*
- *A weighting function (optional)*
- *How to fit with the local points?*

Slide credit: Carlos Guestrin

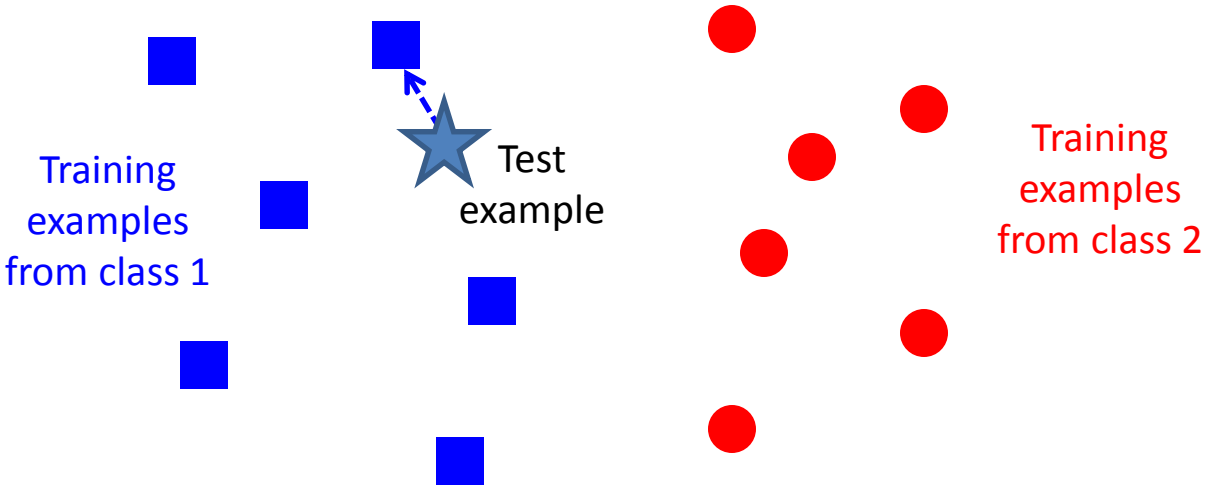
1-Nearest Neighbor Classifier

Four things make a memory based learner:

- *A distance metric*
  - **Euclidean (and others)**
- *How many nearby neighbors to look at?*
  - **1**
- *A weighting function (optional)*
  - **Not used**
- *How to fit with the local points?*
  - **Predict the same output as the nearest neighbor**

Slide credit: Carlos Guestrin

1-Nearest Neighbor Classifier



f(**x**) = label of the training example nearest to **x**

Adapted from Lana Lazebnik

K-Nearest Neighbor Classifier

Four things make a memory based learner:

- *A distance metric*
  - **Euclidean (and others)**
- *How many nearby neighbors to look at?*
  - **K**
- *A weighting function (optional)*
  - **Not used**
- *How to fit with the local points?*
  - **Predict the average output among the nearest neighbors**

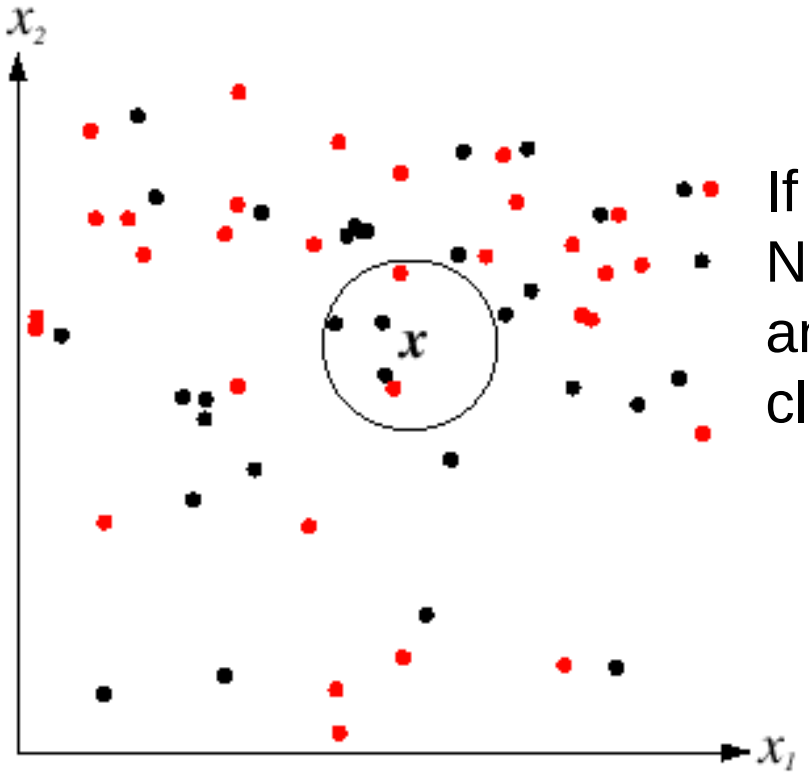
Slide credit: Carlos Guestrin

K-Nearest Neighbor Classifier

- For a new point, find the *k* closest points from training data (e.g. *k*=5)
- Labels of the *k* points "vote" to classify

Black = negative

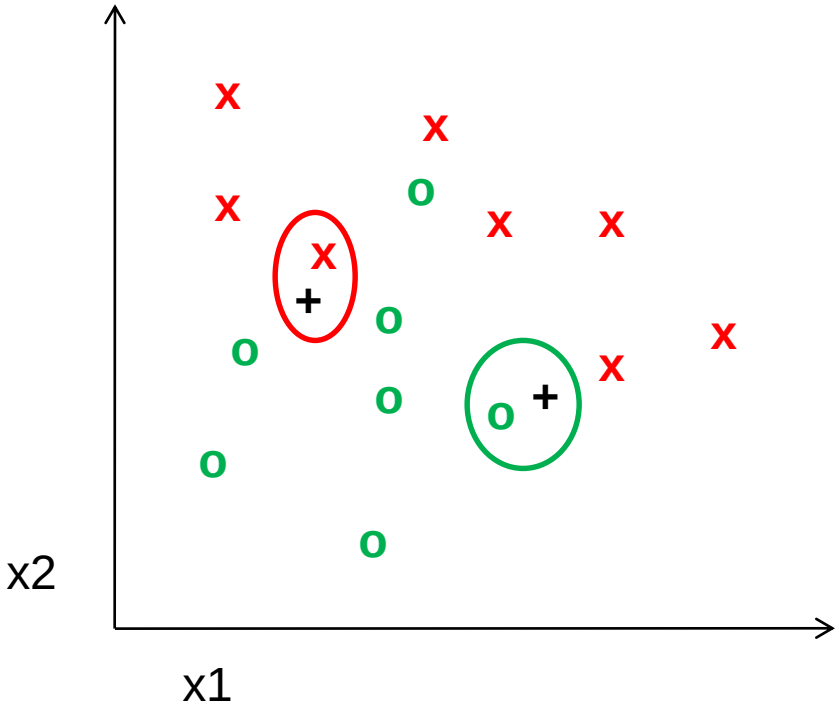
Red = positive



If query lands here, the 5 NN consist of 3 negatives and 2 positives, so we classify it as negative.

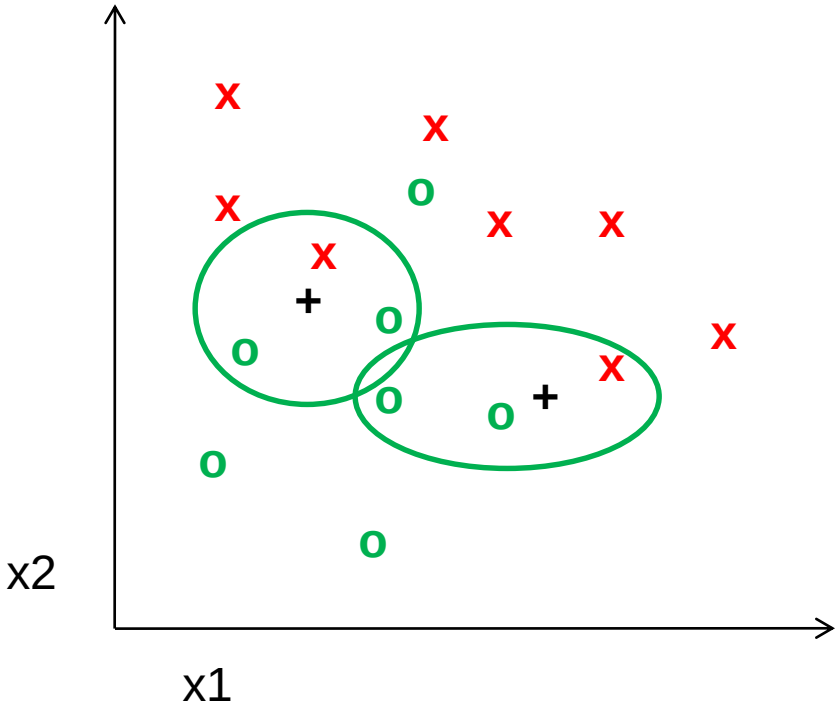
Slide credit: David Lowe

1-nearest neighbor



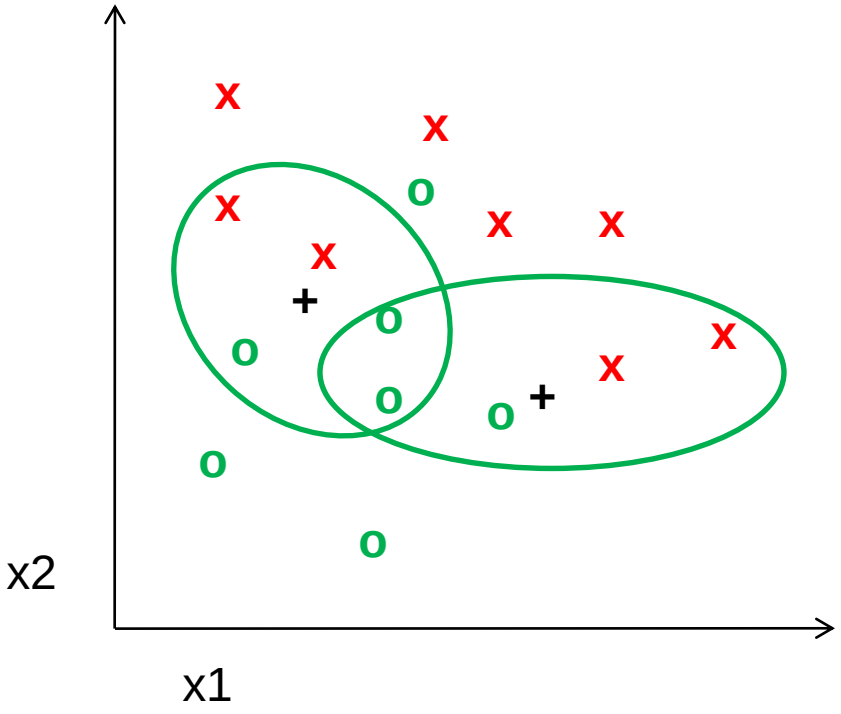
Slide credit: Derek Hoiem

3-nearest neighbor



Slide credit: Derek Hoiem

5-nearest neighbor



What are the tradeoffs of having a too large *k*? Too small *k*?

Slide credit: Derek Hoiem

Formal Definition

- Let *x* be our test data point, and $N_K(x)$ be the indices of the k nearest neighbors of *x*
- Classification:

$$y = \text{argmax}_c \#(y_i = c)$$

$$y = \text{argmax}_c \sum_{i \in N_K(x)} \text{I}(y_i = c)$$

- Regression:

$$y = \frac{1}{K} \sum_{i \in N_K(x)} y_i$$

Example: Predict where this picture was taken

Hays and Efros, **IM2GPS: Estimating Geographic Information from a Single Image**, CVPR 2008

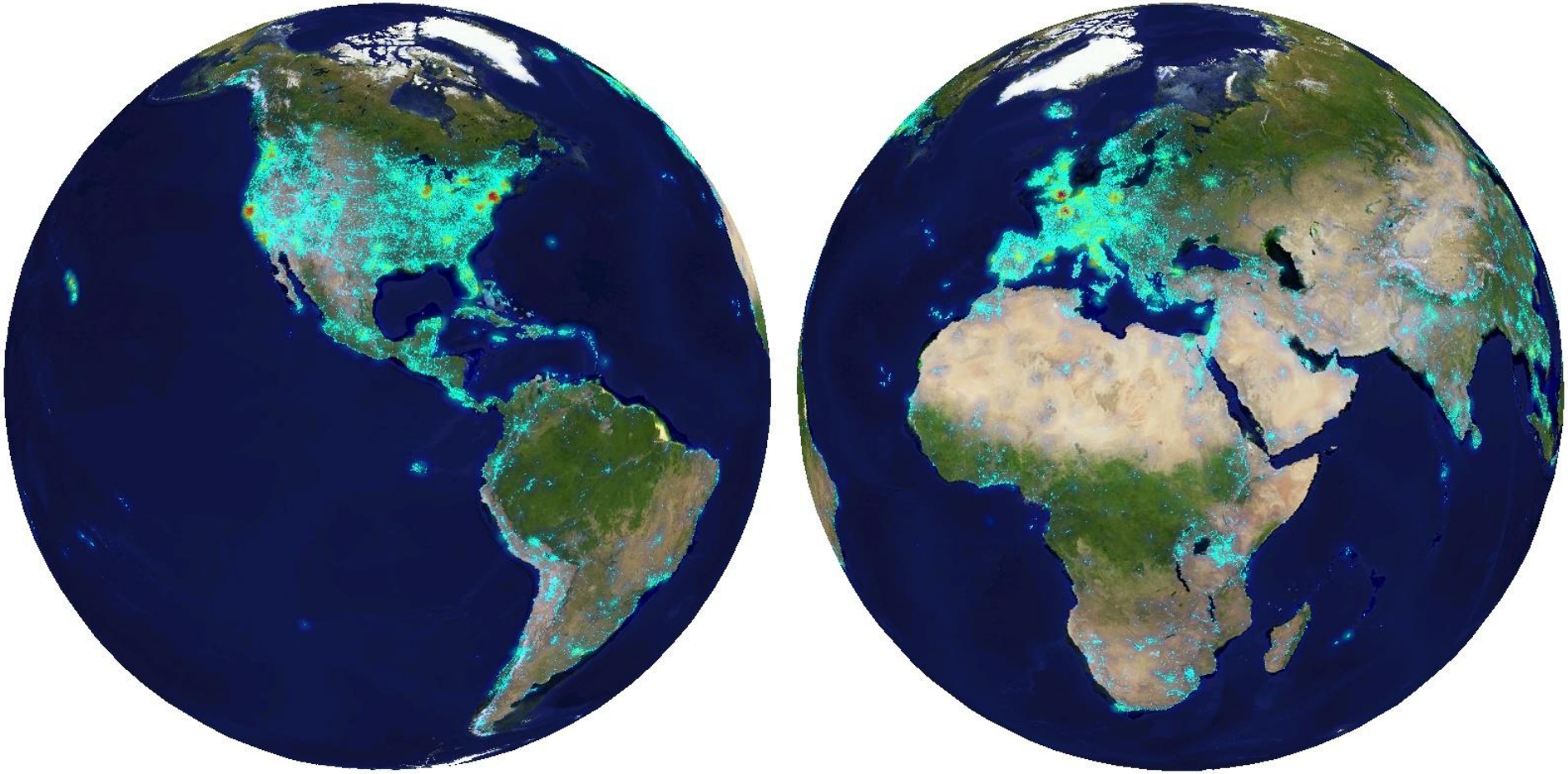
Example: Predict where this picture was taken

Hays and Efros, **IM2GPS: Estimating Geographic Information from a Single Image**, CVPR 2008

Example: Predict where this picture was taken

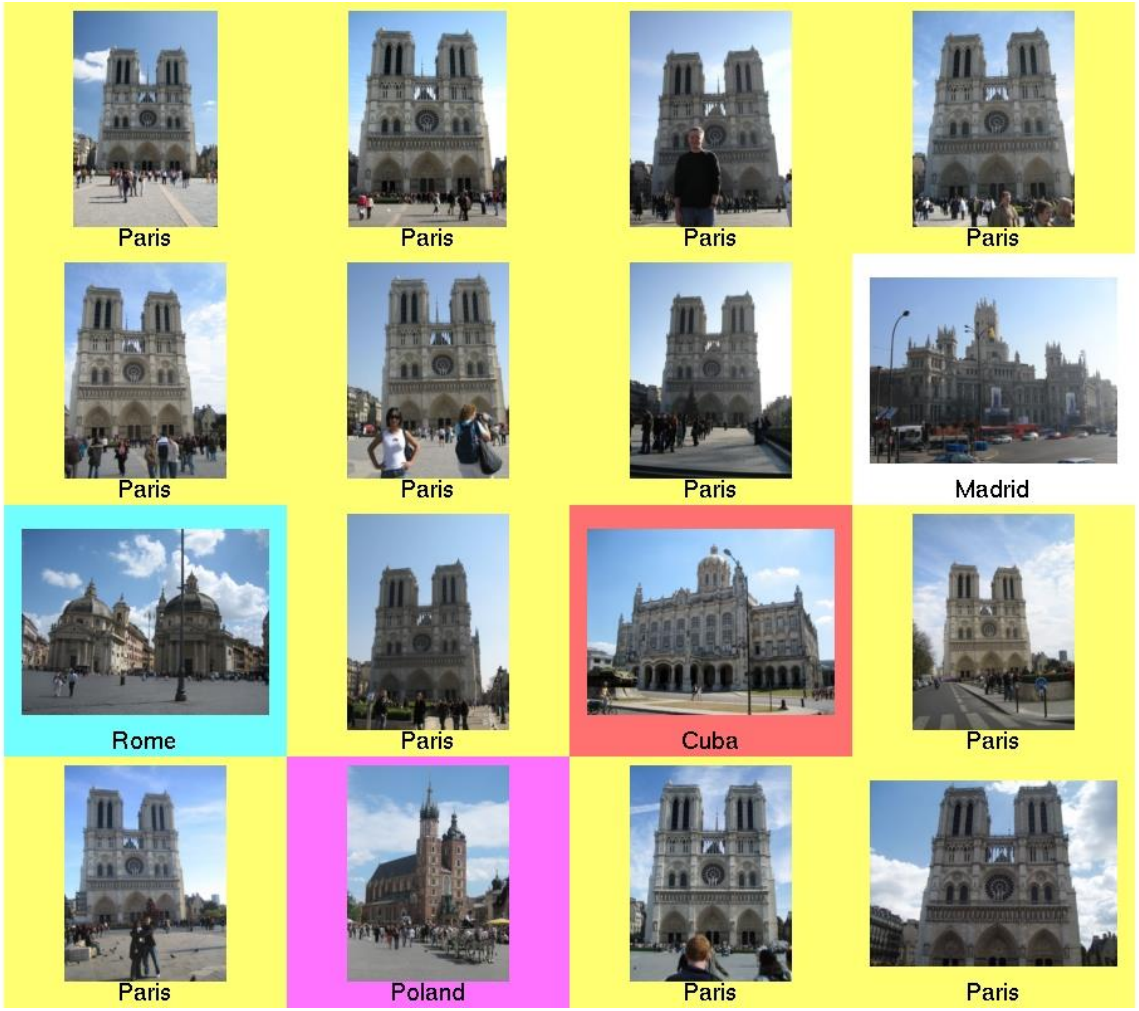
Hays and Efros, **IM2GPS: Estimating Geographic Information from a Single Image**, CVPR 2008

6+ million geotagged photos by 109,788 photographers

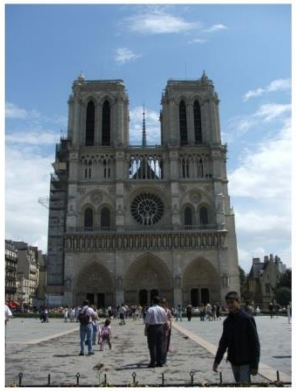


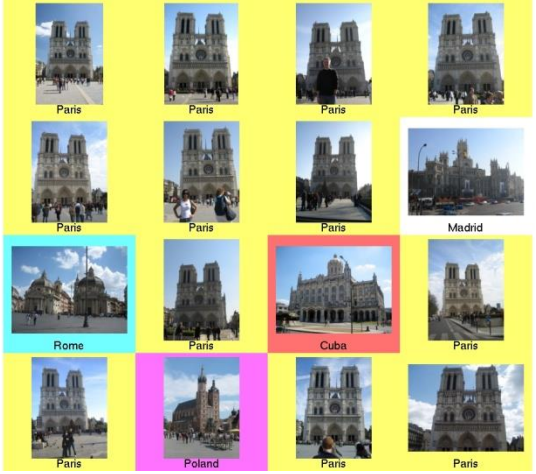
Hays and Efros, **IM2GPS: Estimating Geographic Information from a Single Image**, CVPR 2008

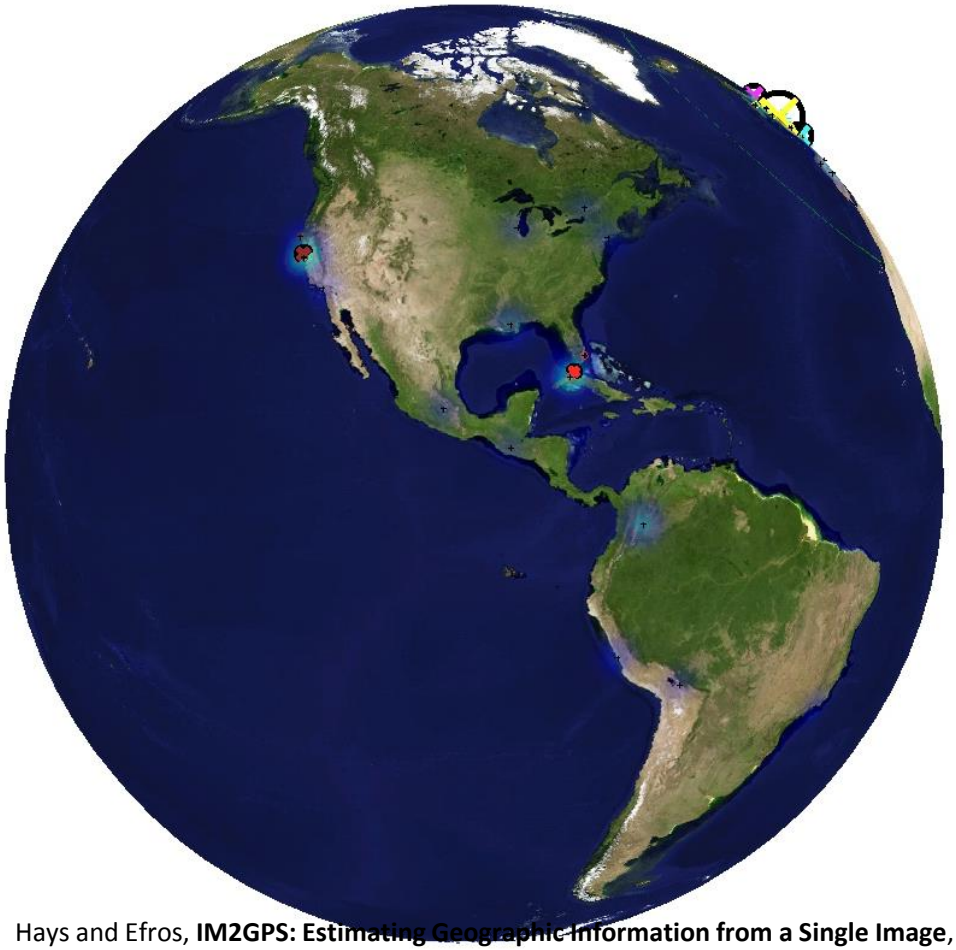
Scene Matches

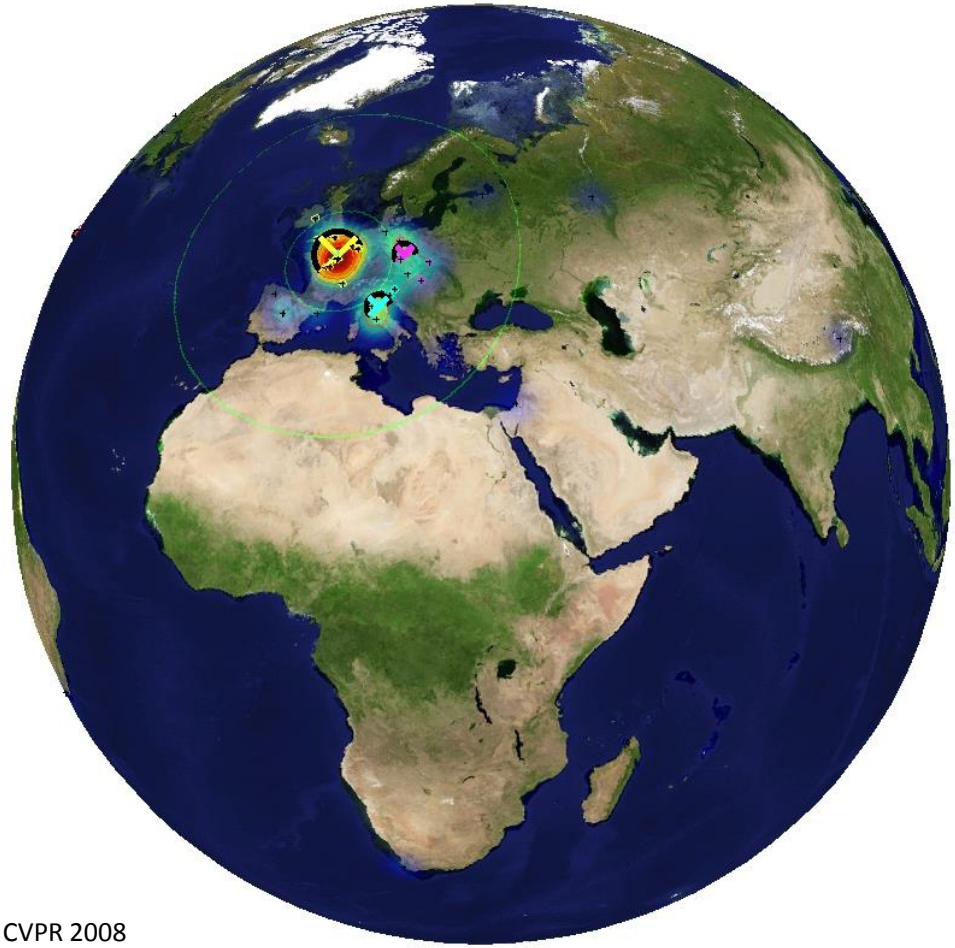


Hays and Efros, **IM2GPS: Estimating Geographic Information from a Single Image**, CVPR 2008









Hays and Efros, **IM2GPS: Estimating Geographic Information from a Single Image**, CVPR 2008

Scene Matches


Madrid


england


France


Paris


Croatia


heidelberg


Macau


Malta


Cairo


Italy


Italy


Italy


Latvia


europe


Barcelona


Austria

Hays and Efros, **IM2GPS: Estimating Geographic Information from a Single Image**, CVPR 2008

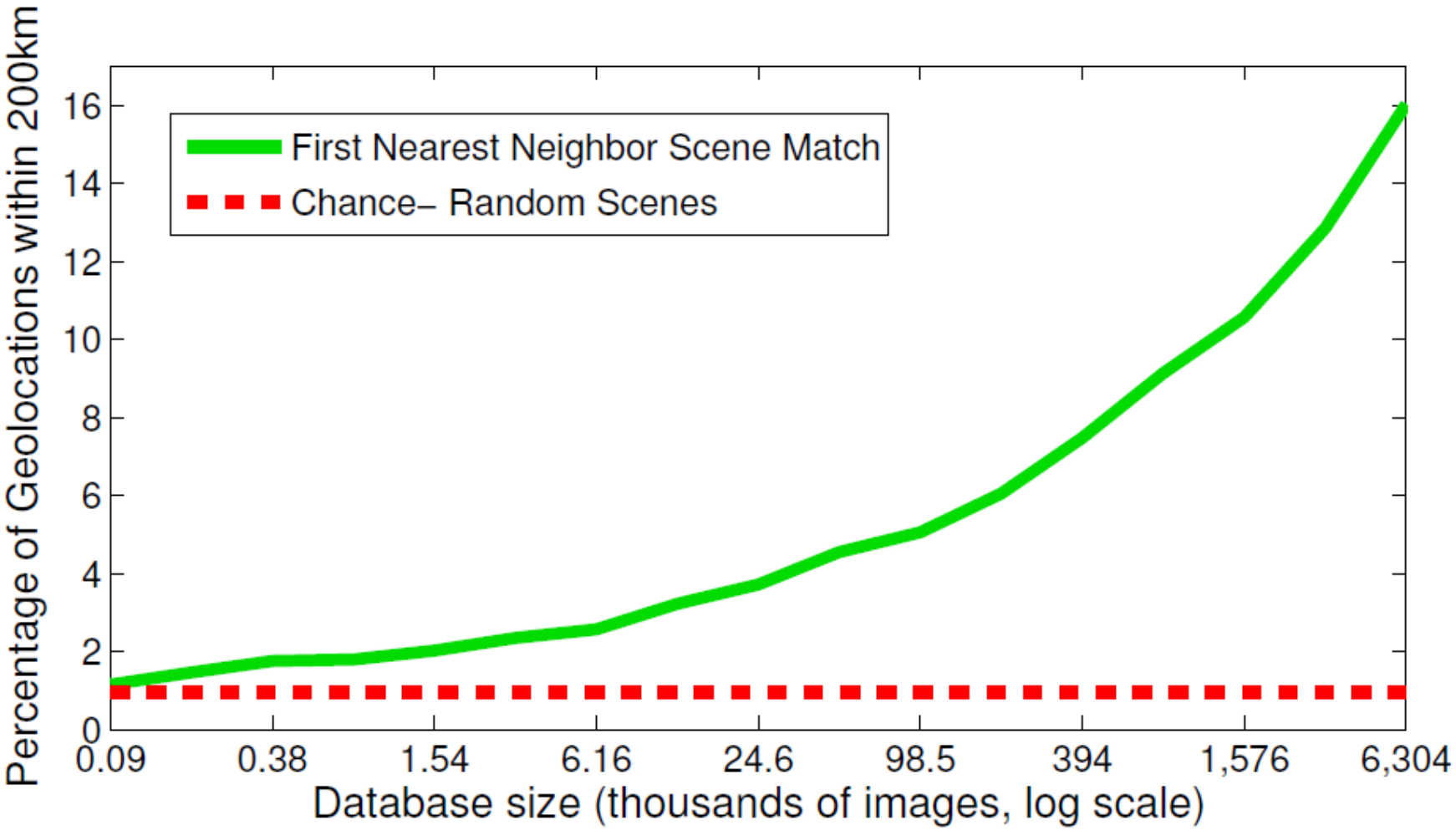
Hays and Efros, **IM2GPS: Estimating Geographic Information from a Single Image**, CVPR 2008

Scene Matches

Hays and Efros, **IM2GPS: Estimating Geographic Information from a Single Image**, CVPR 2008

Hays and Efros, **IM2GPS: Estimating Geographic Information from a Single Image**, CVPR 2008

The Importance of Data



Hays and Efros, **IM2GPS: Estimating Geographic Information from a Single Image**, CVPR 2008

k-Nearest Neighbor

Four things make a memory based learner:

- *A distance metric*
  - **Euclidean (and others)**

- *How many nearby neighbors to look at?*
  - **k**

- *A weighting function (optional)*
  - **Not used**

- *How to fit with the local points?*
  - **Just predict the average output among the nearest neighbors**

Slide credit: Carlos Guestrin

Distances

- Suppose I want to charge my overall distance more for differences in x_2 direction as opposed to x_1 direction
- Setup A: equal weighing on all directions
- Setup B: more weight on x_2 direction
- Will my neighborhoods be longer in the x_1 or x_2 direction?

Voronoi partitioning

- Nearest neighbor regions
- All points in a region are closer to the seed in that region than to any other seed (black dots = seeds)

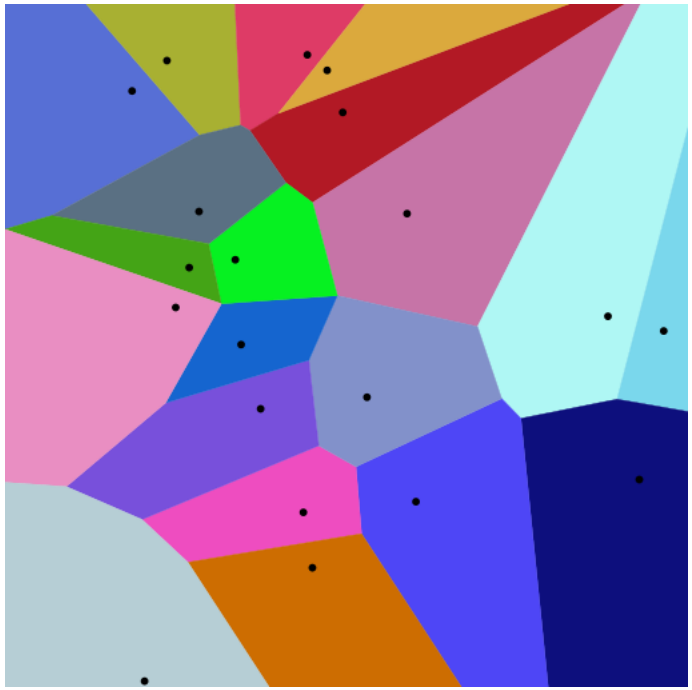
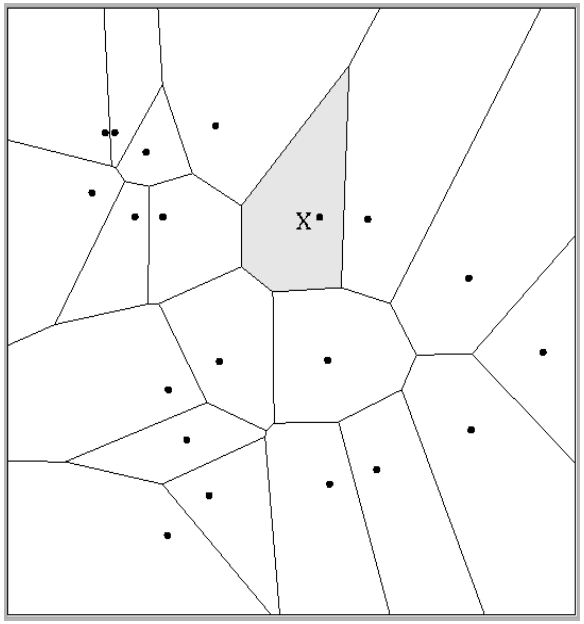


Figure from Wikipedia

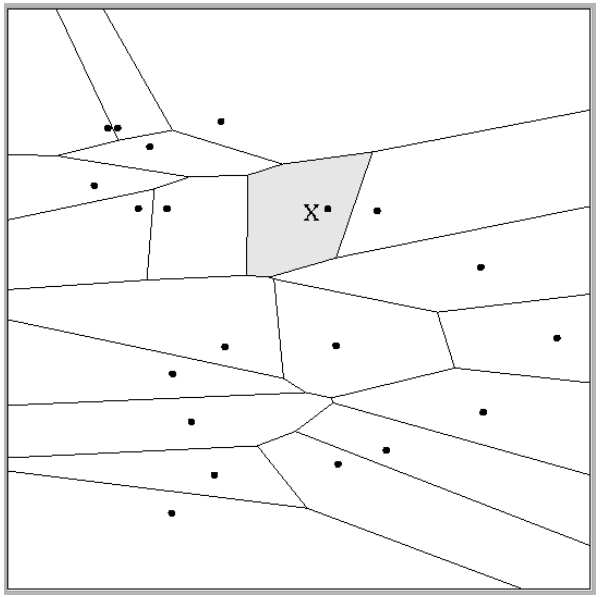
Multivariate distance metrics

Suppose the input vectors $\mathbf{x}_1$, $\mathbf{x}_2$, …$\mathbf{x}_N$ are two dimensional:

$\mathbf{x}_1 = (x_1^1 , x_1^2) , \mathbf{x}_2 = (x_2^1 , x_2^2) , \ldots , \mathbf{x}_N = (x_N^1 , x_N^2)$.



$Dist(\mathbf{x}_i,\mathbf{x}_j) = (x_i^1 - x_j^1)^2 + (x_i^2 - x_j^2)^2$



$Dist(\mathbf{x}_i,\mathbf{x}_j) = (x_i^1 - x_j^1)^2 + (3x_i^2 - 3x_j^2)^2$

The relative scalings in the distance metric affect region shapes

Adapted from Carlos Guestrin

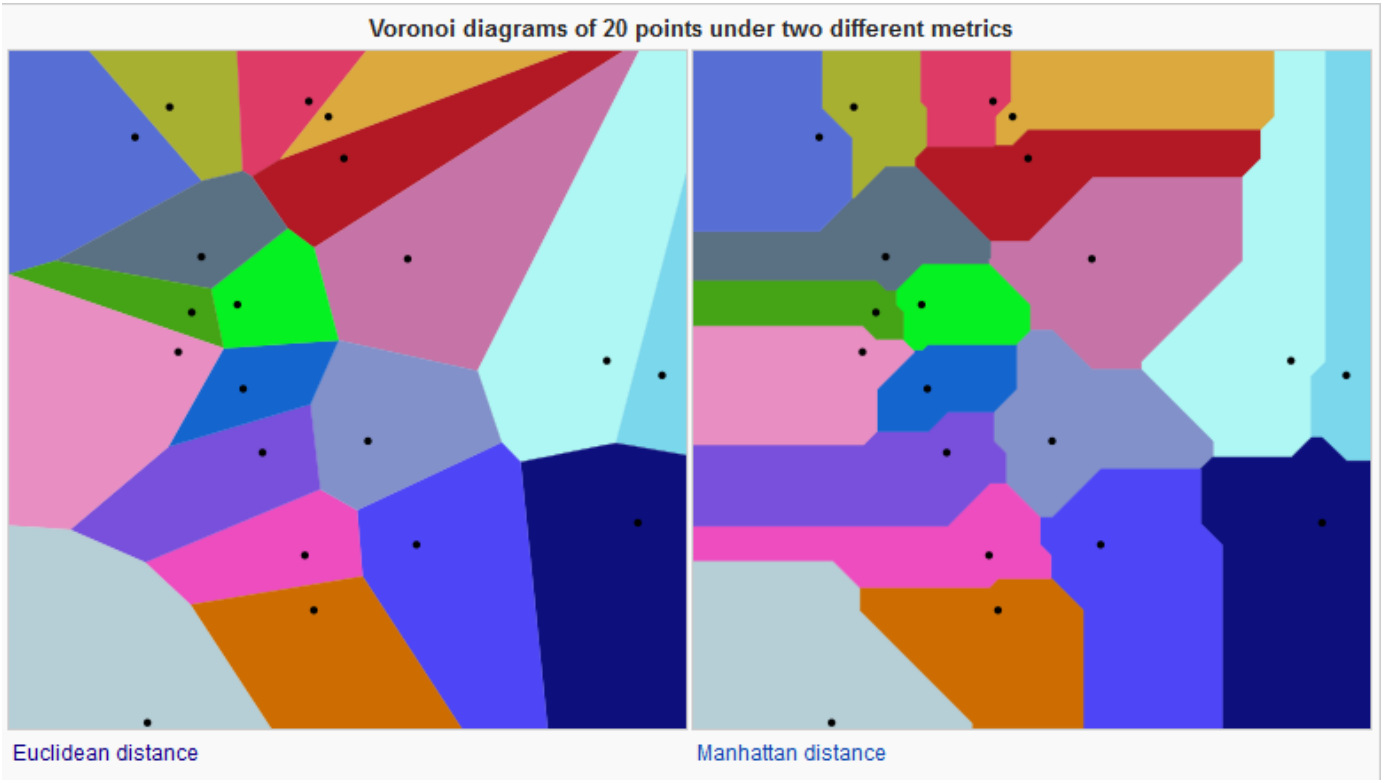
Distance metrics

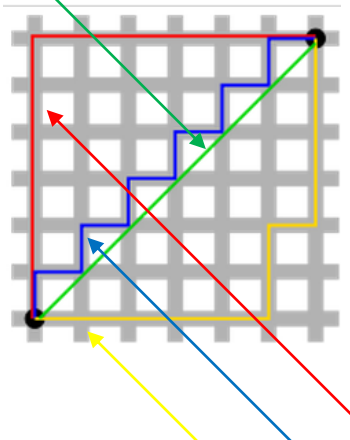
- Euclidean: $$d(\mathbf{x}, \mathbf{z}) = [\sum_{i=1}^{D} (x_i - z_i)^2]^{\frac{1}{2}}$$
- Minkowski: $$d(\mathbf{x}, \mathbf{z}) = [\sum_{i=1}^{D} |x_i - z_i|^P]^{\frac{1}{P}}$$
- Mahalanobis: $$d(\mathbf{x}, \mathbf{z}) = \sqrt{\sum_{i=1}^{D} \frac{(x_i - z_i)^2}{\sigma_i^2}}$$

$$d(\mathbf{x}, \mathbf{z}) = (\mathbf{x} - \mathbf{z})^{\mathrm{T}} A (\mathbf{x} - \mathbf{z})$$

(where *A* is a **positive semidefinite** matrix,
i.e. symmetric matrix with all non-negative eigenvalues)

Distance metrics





Figures from Wikipedia

Another generalization: Weighted K-NNs

- Neighbors weighted differently:
  - Use all samples, i.e. K = N
  - Weight on i-th sample: $w_i = e^{\frac{-||\mathbf{x}-\mathbf{x}_i||^2}{\sigma^2}}$
  - σ = the bandwidth parameter, expresses how quickly our weight function "drops off" as points get further and further from the query x

- Classification: $y = \text{argmax}_c \sum_{i=1}^{N} w_i \text{I}(y_i = c)$

- Regression: $y = \frac{\sum_{i=1}^{N} w_i y_i}{\sum_{i=1}^{N} w_i}$

Another generalization: Weighted K-NNs

- Extremes
  - Bandwidth = infinity: prediction is dataset average
  - Bandwidth = zero: prediction becomes 1-NN

Kernel Regression/Classification

Four things make a memory based learner:

- *A distance metric*
  - **Euclidean (and others)**

- *How many nearby neighbors to look at?*
  - **All of them**

- *A weighting function (optional)*
  - $w_i = exp(-d(x_i, query)^2 / \sigma^2)$
  - Nearby points to the query are weighted strongly, far points weakly. The σ parameter is the **kernel width / bandwidth**.

- *How to fit with the local points?*
  - **Predict the weighted average of the outputs**

Adapted from Carlos Guestrin

Problems with Instance-Based Learning

- Too many features?
  - Doesn't work well if large number of irrelevant features, distances overwhelmed by noisy features
  - Distances become meaningless in high dimensions (the **curse of dimensionality**)

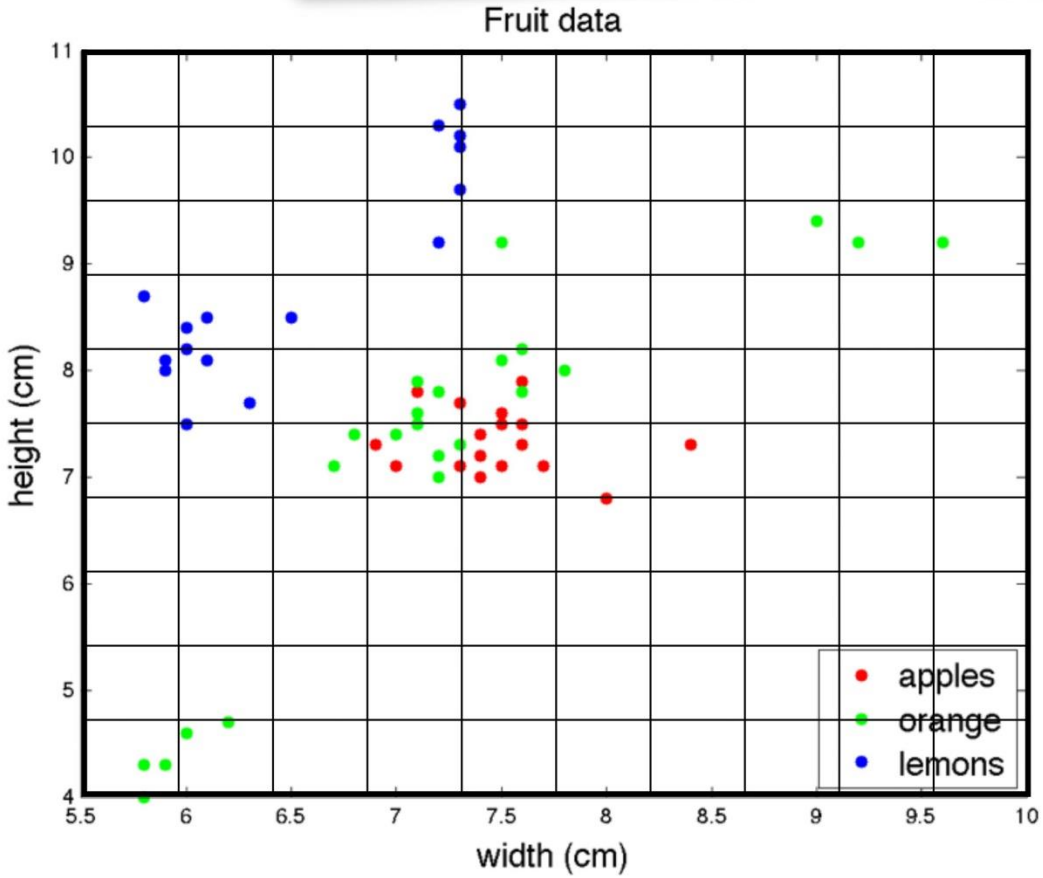
- What is the impact of the **value of K**?

- Expensive
  - No learning: most real work done during testing
  - For every test sample, must search through all dataset – very slow!
  - **Must use tricks like approximate nearest neighbor search**
  - Need to store all training data

Adapted from Dhruv Batra

Curse of Dimensionality

How many neighborhoods are there?



#bins = 10x10

d = 2

#bins = 10^{d}

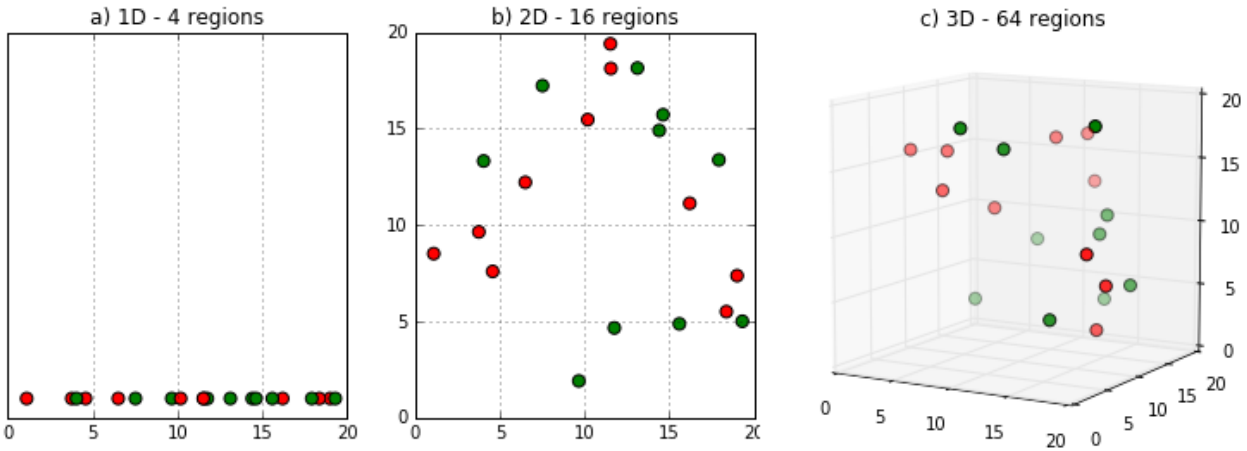
d = 1000

Atoms in the universe

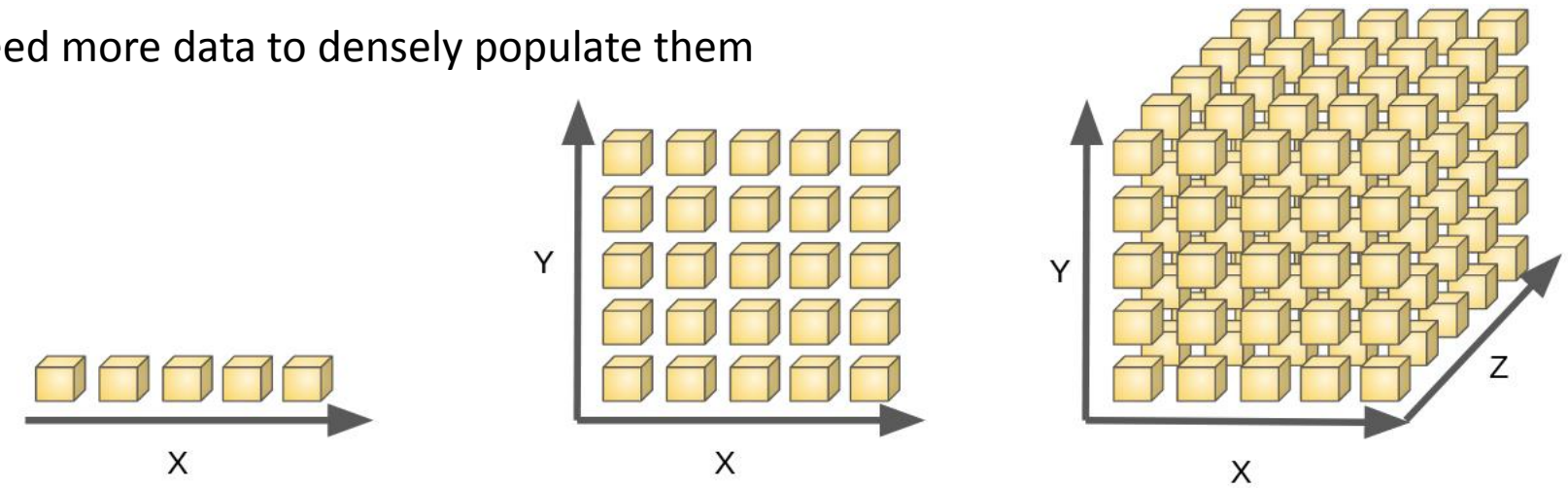
~ 10^{80}

Curse of Dimensionality

Regions become more sparsely populated given the same amount of data



Need more data to densely populate them



Figures from https://www.kdnuggets.com/2017/04/must-know-curse-dimensionality.html, https://medium.freecodecamp.org/the-curse-of-dimensionality-how-we-can-save-big-data-from-itself-d9fa0f872335

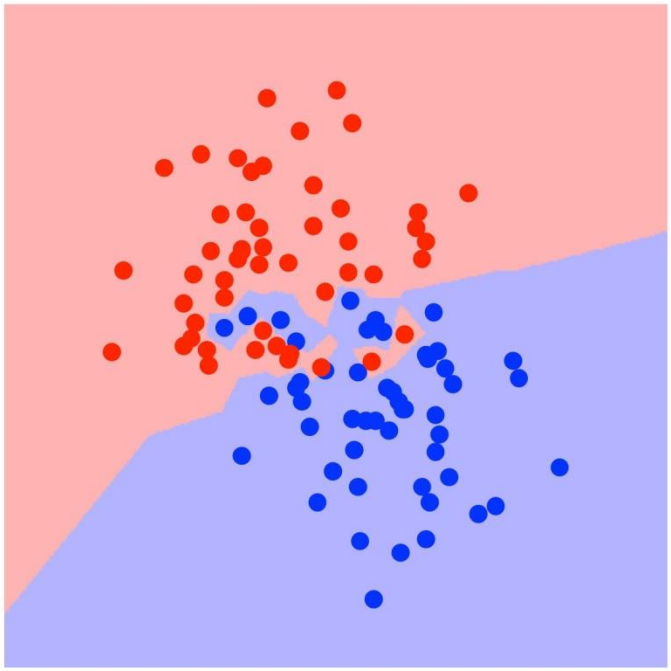
kNN Decision Boundary

- Increasing k simplifies / complicates decision boundary

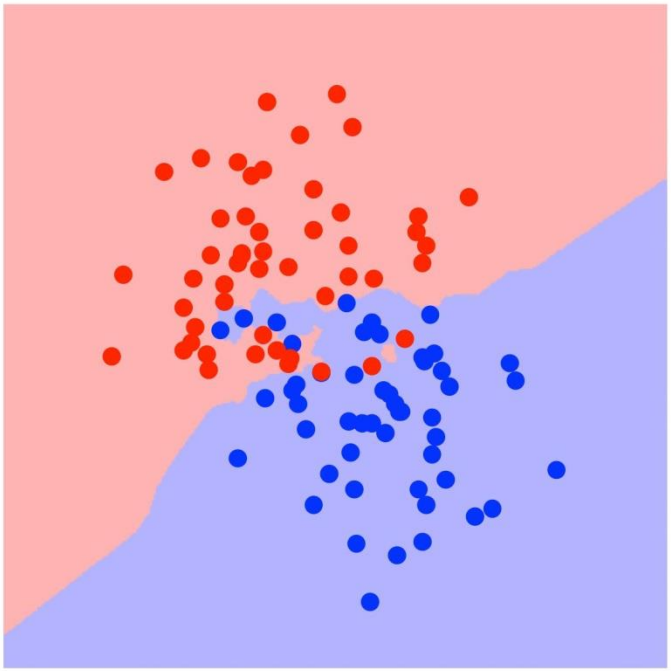
kNN Decision Boundary

- Increasing k “simplifies” decision boundary
  - Majority voting means less emphasis on individual points

K = 1



K = 3

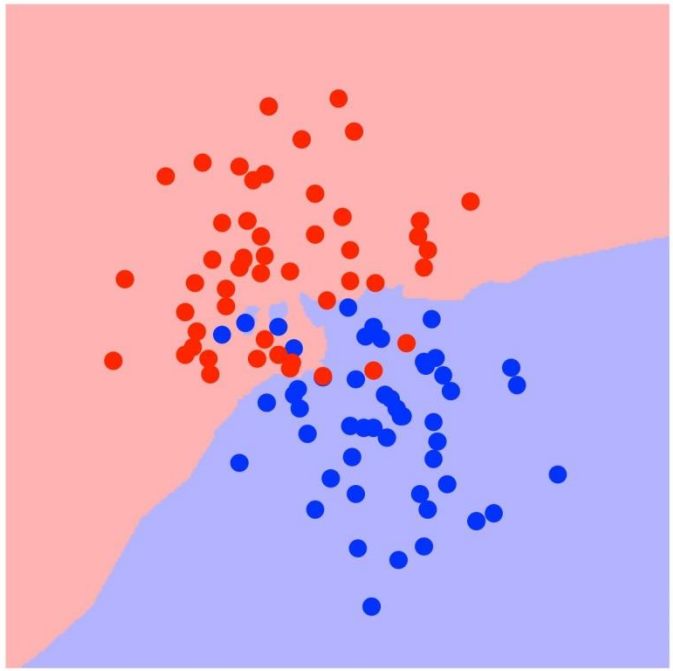


Slide credit: Alexander Ihler

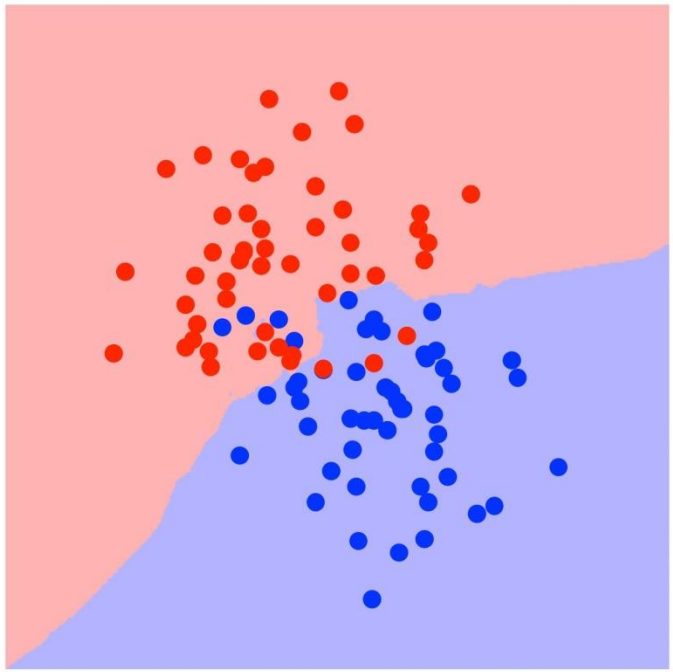
kNN Decision Boundary

- Increasing k “simplifies” decision boundary
  - Majority voting means less emphasis on individual points

K = 5



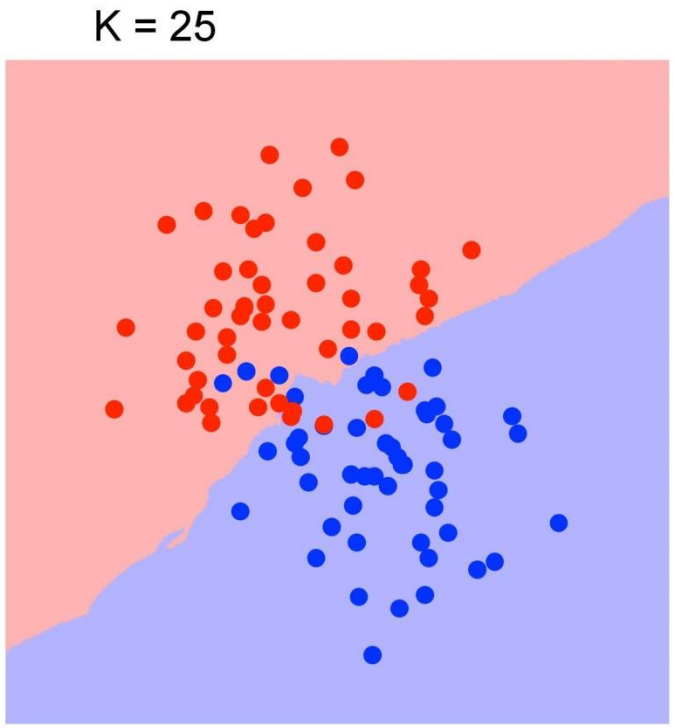
K = 7



Slide credit: Alexander Ihler

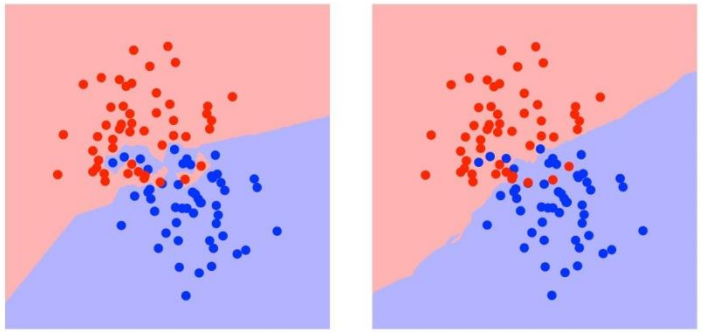
kNN Decision Boundary

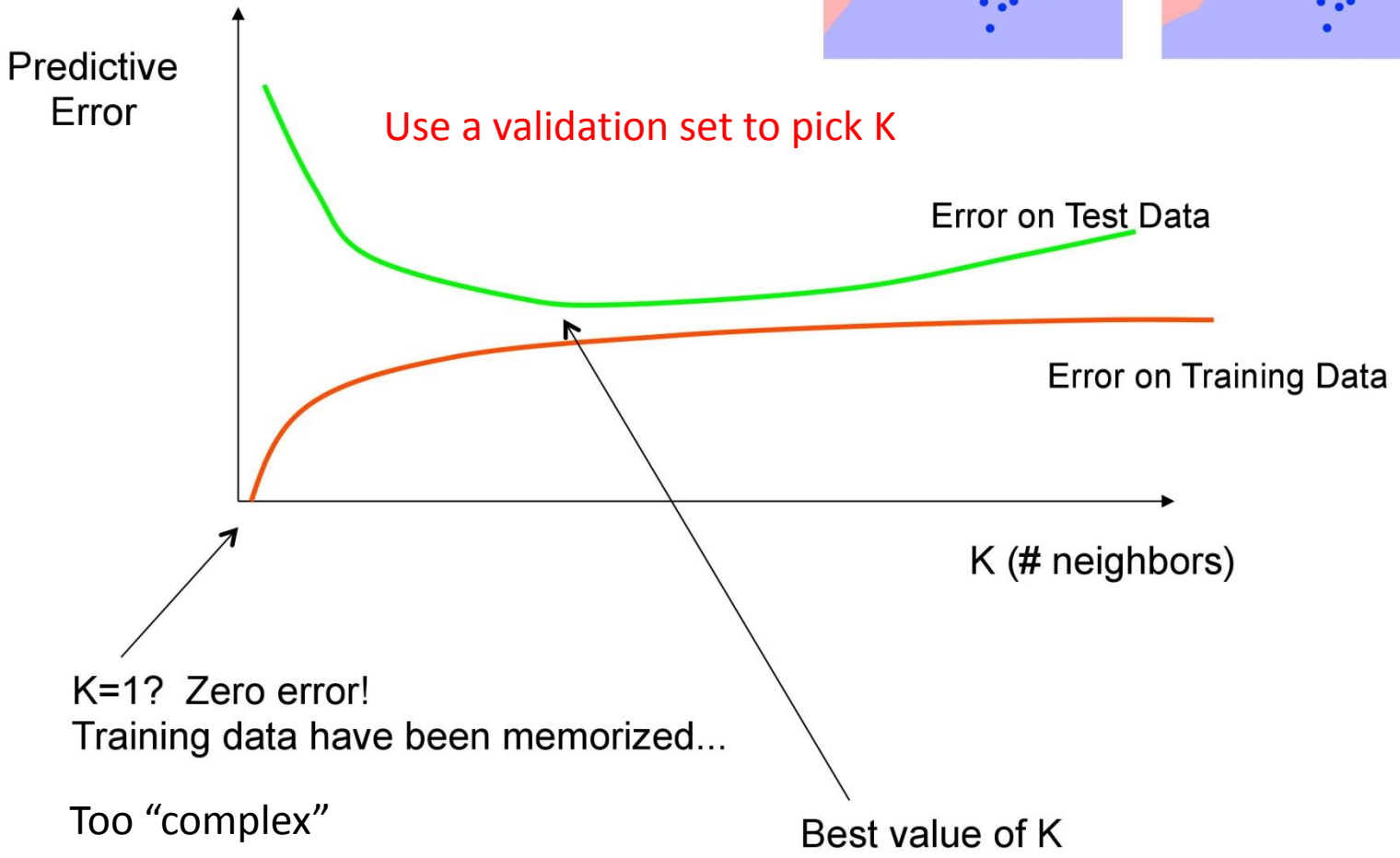
- Increasing k “simplifies” decision boundary
  - Majority voting means less emphasis on individual points



Slide credit: Alexander Ihler

Error rates and K





Slide credit: Alexander Ihler

Summary

- K-Nearest Neighbor is the most basic and simplest to implement classifier
- Cheap at training time, expensive at test time
- Unlike other methods we'll see later, naturally works for any number of classes
- Pick K through a validation set, use approximate methods for finding neighbors
- Success of classification depends on the amount of data and the meaningfulness of the distance function (also true for other algorithms)

Plan for this lecture

- The simplest classifier: K-Nearest Neighbors
  - Algorithm and example use
  - Generalizing: Distance metrics, weighing neighbors
  - Problems: curse of dimensionality, picking K
- Logistic regression
  - Probability: review
  - Linear regression for classification?
  - Maximum likelihood solution for *logistic* regression
  - Related algorithm: perceptron

Probability Review

A is non-deterministic event

Can think of A as a Boolean-valued variable

Examples

A = your next patient has cancer

A = Steelers win Super Bowl LIII

Interpreting Probabilities

What does P(A) mean?

Frequentist View

limit N→∞ #(A is true)/N

frequency of a repeating non-deterministic event

Bayesian View

P(A) is your "belief" about A

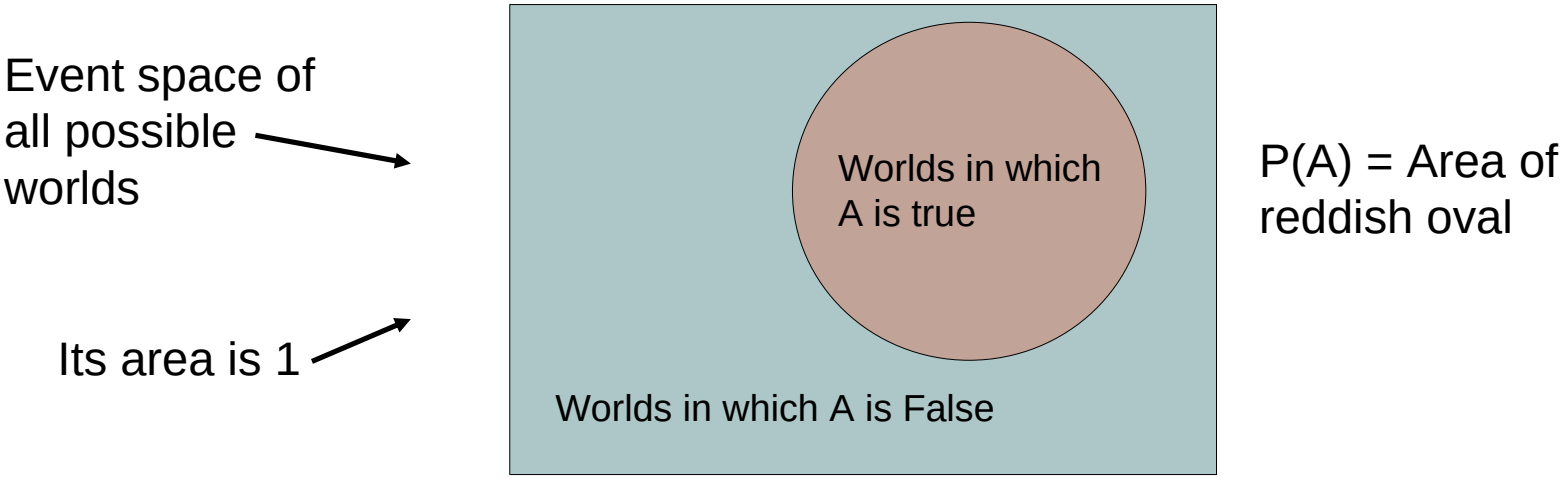
Axioms of Probability

$0 <= P(A) <= 1$

$P(\text{false}) = 0$

$P(\text{true}) = 1$

$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$



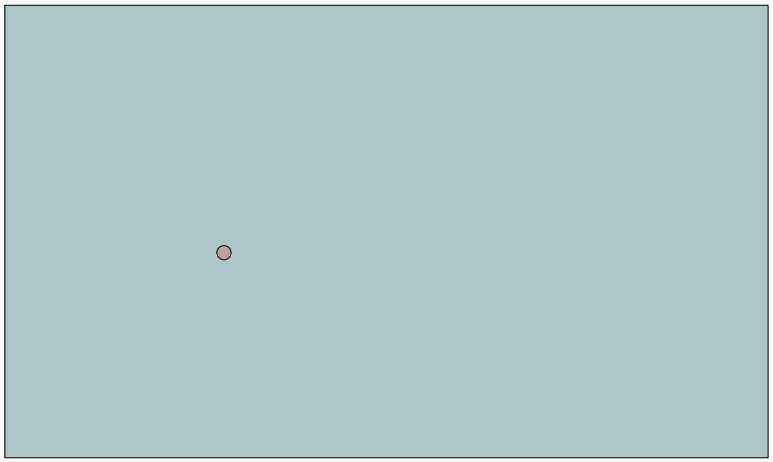
Axioms of Probability

$0 <= P(A) <= 1$

$P(false) = 0$

$P(true) = 1$

$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$



The area of A can't get any smaller than 0

And a zero area would mean no world could ever have A true

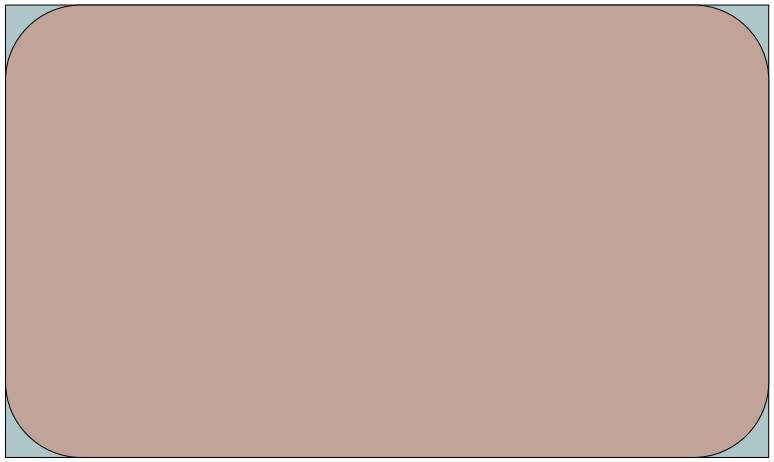
Axioms of Probability

$0 <= P(A) <= 1$

$P(false) = 0$

$P(true) = 1$

$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$



The area of A can't get any bigger than 1

And an area of 1 would mean all worlds will have A true

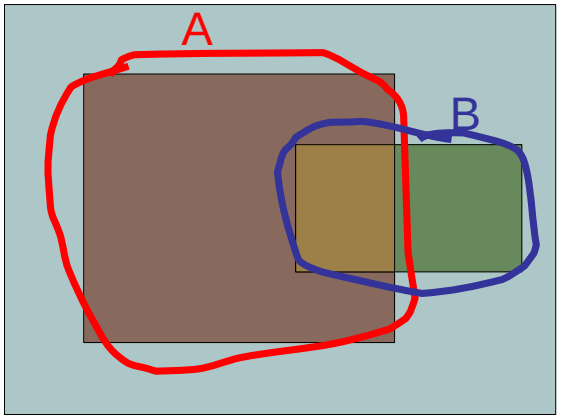
Axioms of Probability

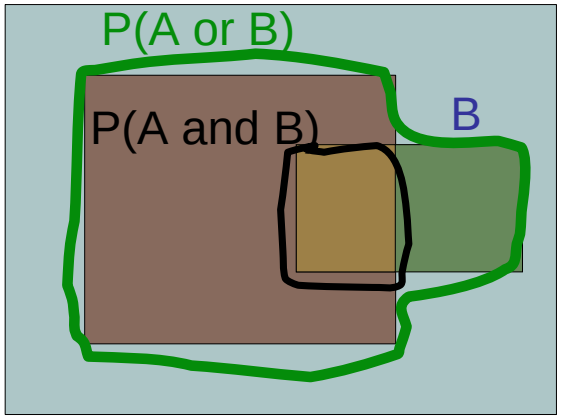
$0 <= P(A) <= 1$

$P(false) = 0$

$P(true) = 1$

$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$

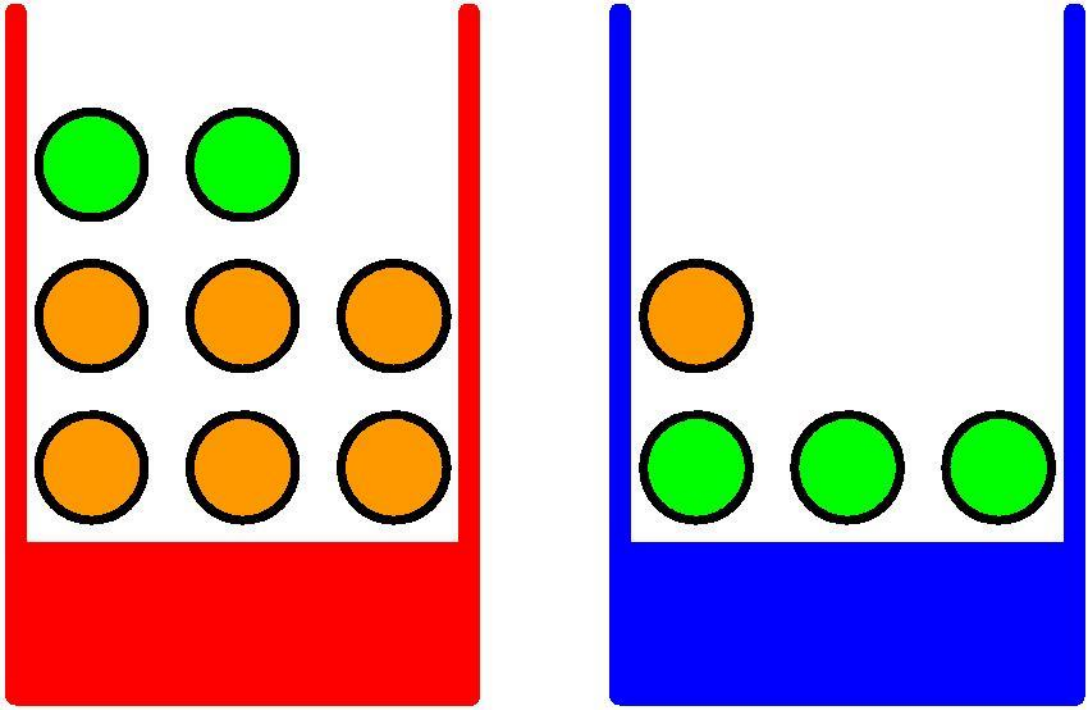




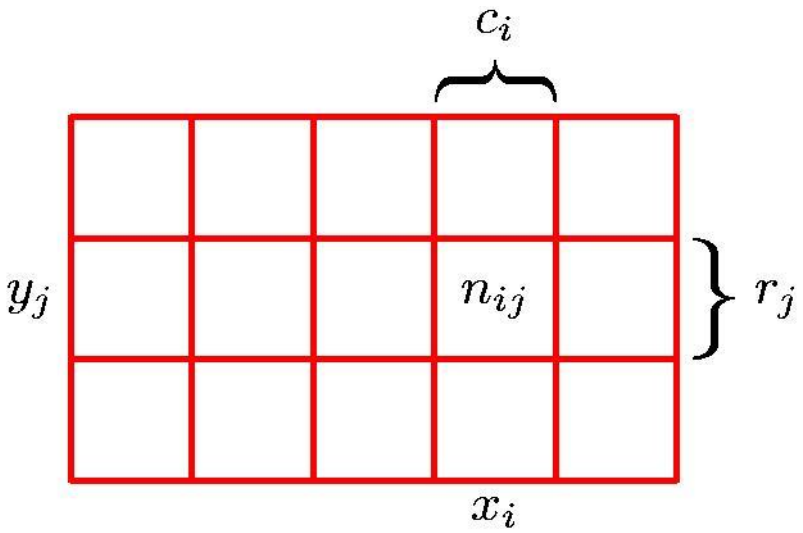
Simple addition and subtraction

Probabilities: Example Use

Apples and Oranges



Marginal, Joint, Conditional



Marginal Probability

$$p(X = x_i) = \frac{c_i}{N}.$$

Joint Probability

$$p(X = x_i, Y = y_j) = \frac{n_{ij}}{N}$$

Conditional Probability

$$p(Y = y_j | X = x_i) = \frac{n_{ij}}{c_i}$$

Joint Probability

- P(X_1,…,X_n) gives the probability of every combination of values (an *n*-dimensional array with v^n values if all variables are discrete with *v* values, all v^n values must sum to 1):

positive

	circle	square
red	0.20	0.02
blue	0.02	0.01

negative

	circle	square
red	0.05	0.30
blue	0.20	0.20

- The probability of all possible conjunctions (assignments of values to some subset of variables) can be calculated by summing the appropriate subset of values from the joint distribution.

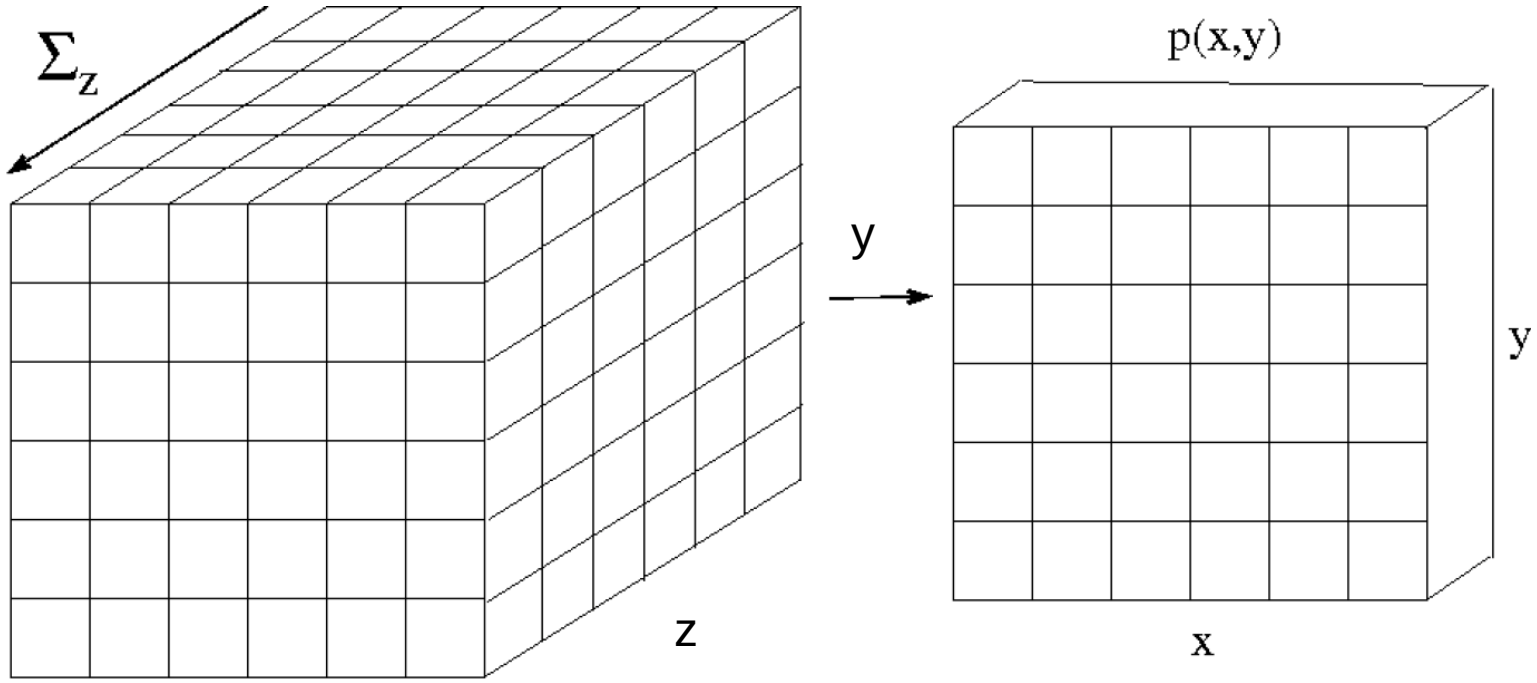
$$P(red \wedge circle)$$
$$P(red)$$

- Therefore, all conditional probabilities can also be calculated.

$$P(positive \mid red \wedge circle)$$

Adapted from Ray Mooney

Marginal Probability



$$p(x,y) = \sum_{z \in \mathcal{Z}} p(x,y,z) \qquad p(x) = \sum_{y \in \mathcal{Y}} p(x,y)$$

Conditional Probability

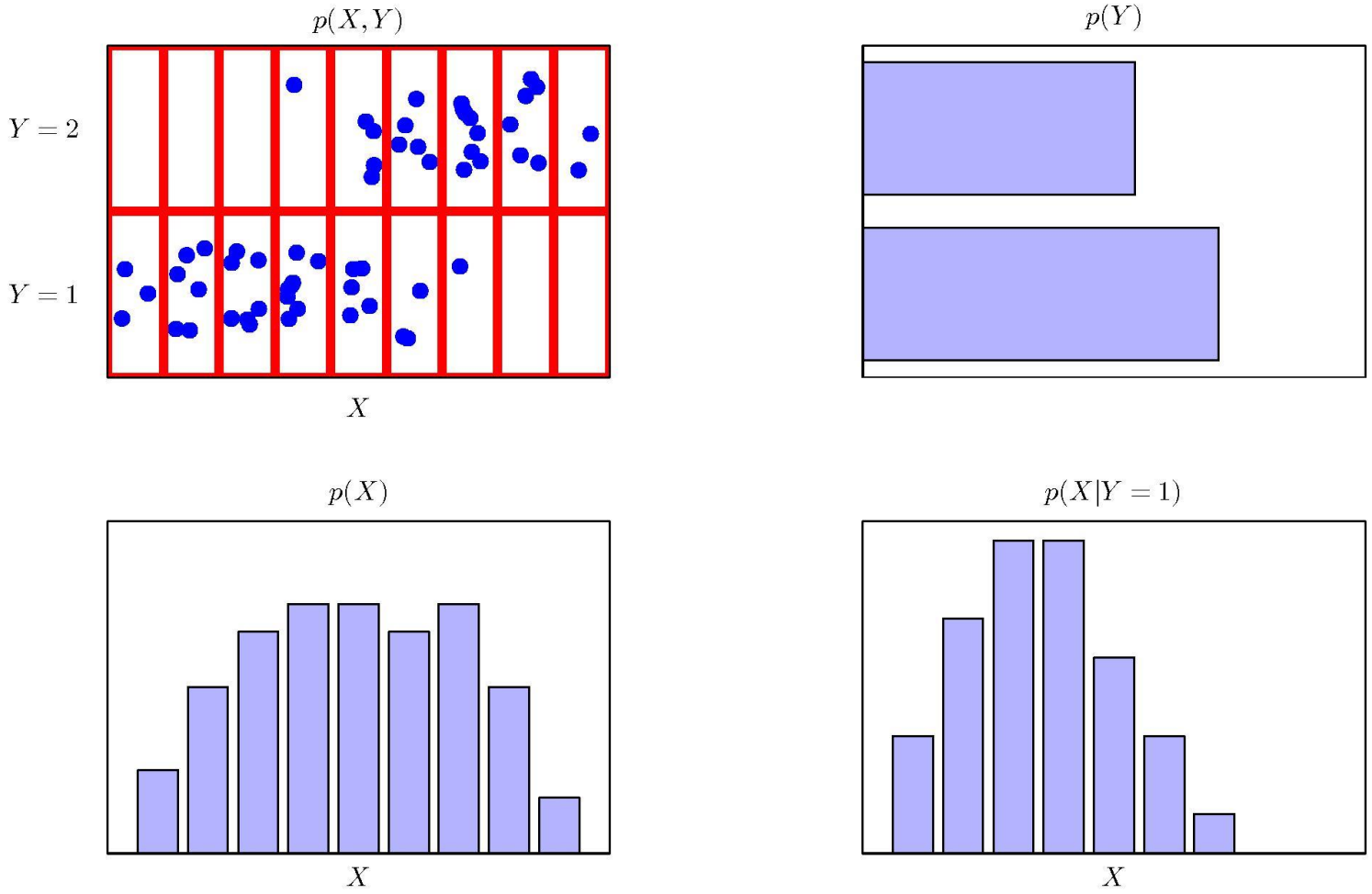
P(Y=y | X=x): What do you believe about Y=y, if I tell you X=x?

P(Andy Murray wins Australian Open 2019)?

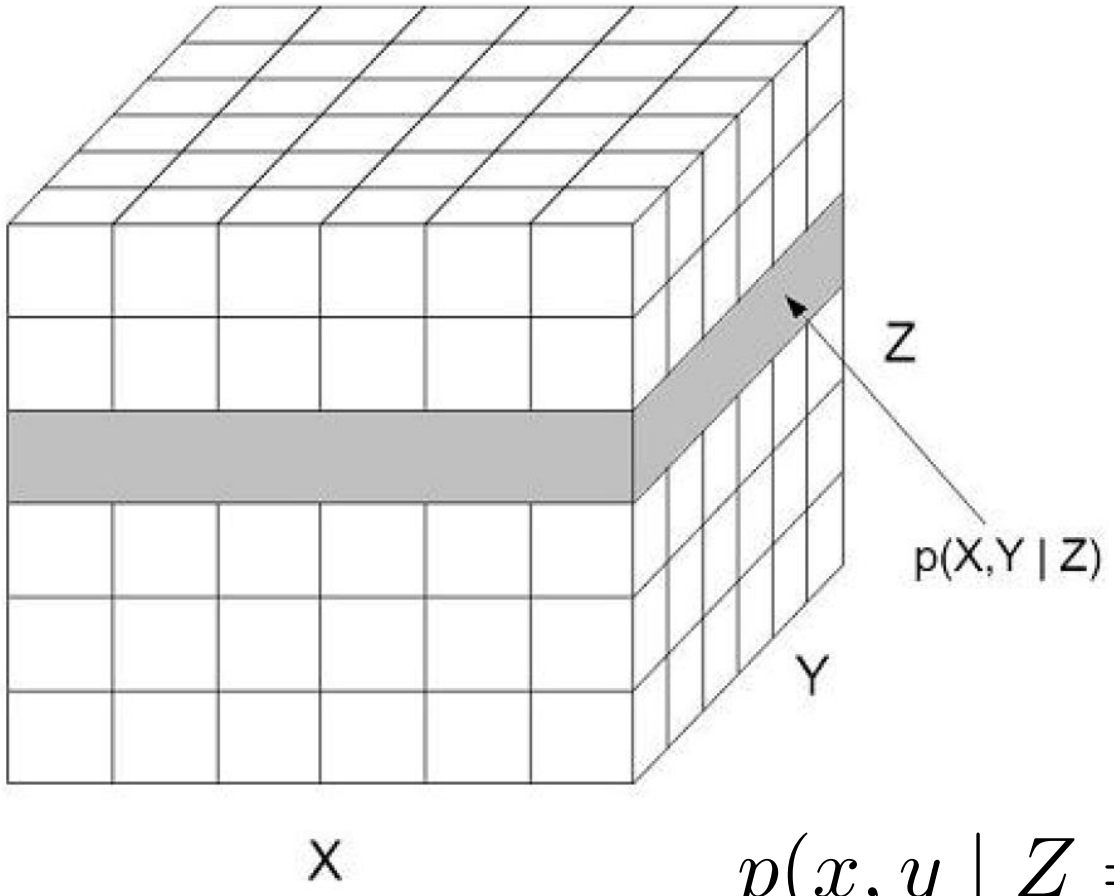
What if I tell you:

- He has won it five times before
- He is currently ranked #307

Conditional Probability

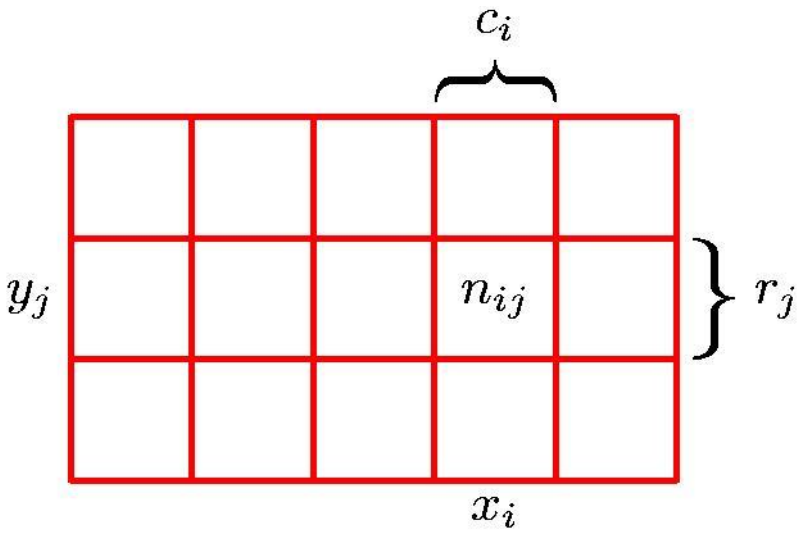


Conditional Probability



$$p(x, y \mid Z = z) = \frac{p(x, y, z)}{p(z)}$$

Sum and Product Rules



Sum Rule

$$p(X = x_i) = \frac{c_i}{N} = \frac{1}{N} \sum_{j=1}^{L} n_{ij}$$

$$= \sum_{j=1}^{L} p(X = x_i, Y = y_j)$$

Product Rule

$$p(X = x_i, Y = y_j) = \frac{n_{ij}}{N} = \frac{n_{ij}}{c_i} \cdot \frac{c_i}{N}$$

$$= p(Y = y_j | X = x_i) p(X = x_i)$$

Chain Rule

Generalizes the product rule:

$$\mathrm{P}\left(\bigcap_{k=1}^{n} A_k\right) = \prod_{k=1}^{n} \mathrm{P}\left(A_k \,\middle|\, \bigcap_{j=1}^{k-1} A_j\right)$$

Example:

$$\mathrm{P}(A_4, A_3, A_2, A_1) = \mathrm{P}(A_4 \mid A_3, A_2, A_1) \cdot \mathrm{P}(A_3 \mid A_2, A_1) \cdot \mathrm{P}(A_2 \mid A_1) \cdot \mathrm{P}(A_1)$$

Equations from Wikipedia

Independence

A and *B* are *independent* iff:

$$P(A|B) = P(A)$$
$$P(B|A) = P(B)$$

These two constraints are logically equivalent

Therefore, if *A* and *B* are independent:

$$P(A|B) = \frac{P(A \wedge B)}{P(B)} = P(A)$$

$$P(A \wedge B) = P(A)P(B)$$

Independence

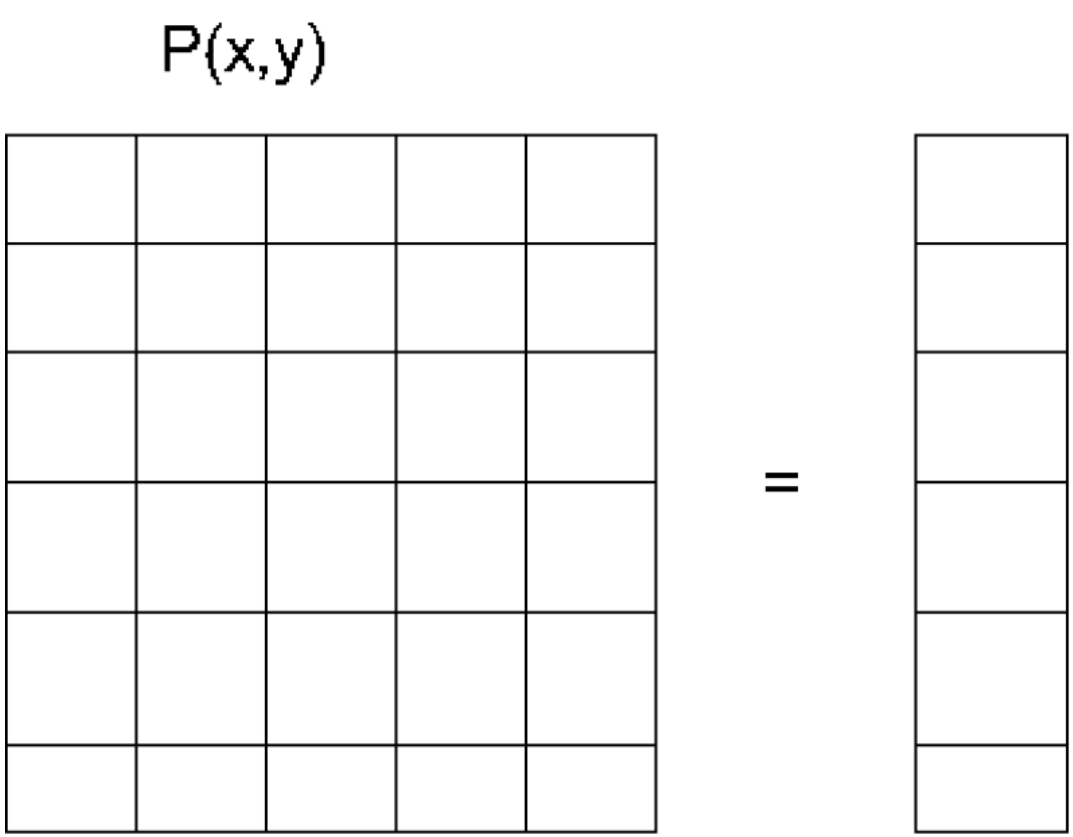
Marginal: *P* satisfies ($\mathbf{X} \perp \mathbf{Y}$) if and only if

$$P(\mathbf{X}=\mathbf{x},\mathbf{Y}=\mathbf{y}) = P(\mathbf{X}=\mathbf{x})\, P(\mathbf{Y}=\mathbf{y}),$$
$$\forall \mathbf{x} \in \text{Val}(\mathbf{X}),\ \mathbf{y} \in \text{Val}(\mathbf{Y})$$

Conditional: *P* satisfies ($\mathbf{X} \perp \mathbf{Y} \mid \mathbf{Z}$) if and only if

$$P(\mathbf{X},\mathbf{Y}|\mathbf{Z}) = P(\mathbf{X}|\mathbf{Z})\, P(\mathbf{Y}|\mathbf{Z}),$$
$$\forall \mathbf{x} \in \text{Val}(\mathbf{X}),\ \mathbf{y} \in \text{Val}(\mathbf{Y}),\ \mathbf{z} \in \text{Val}(\mathbf{Z})$$

Independence



$$X \perp Y$$

$$p(x, y) = p(x)p(y)$$

$$\text{for all } x \in \mathcal{X}, y \in \mathcal{Y}$$

Bayes' Theorem

$$p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)}$$

$$p(X) = \sum_{Y} p(X|Y)p(Y)$$

posterior $\propto$ likelihood × prior

Expectations

$$\mathbb{E}[f] = \sum_x p(x) f(x)$$

$$\mathbb{E}[f] = \int p(x) f(x)\, \mathrm{d}x$$

$$\mathbb{E}_x[f|y] = \sum_x p(x|y) f(x)$$

Conditional Expectation (discrete)

$$\mathbb{E}[f] \simeq \frac{1}{N} \sum_{n=1}^{N} f(x_n)$$

Approximate Expectation (discrete and continuous)

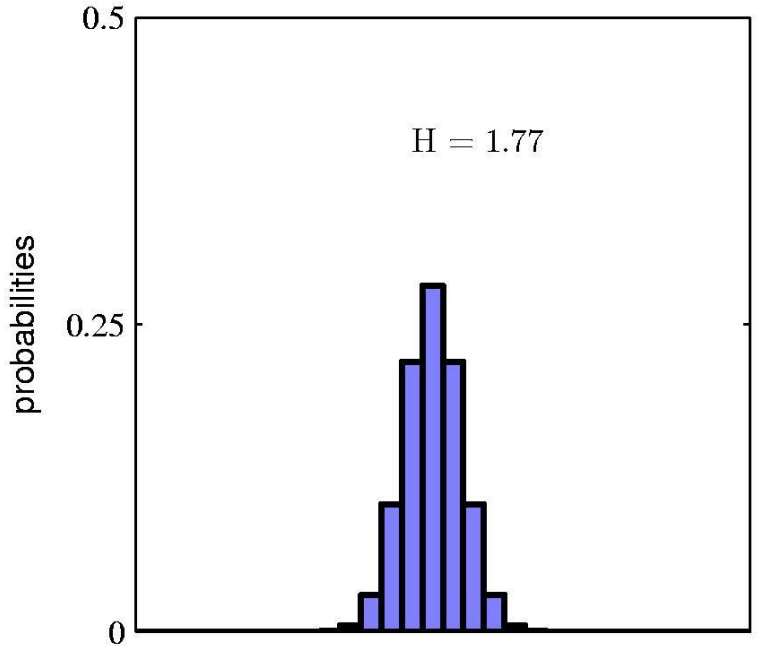
Entropy

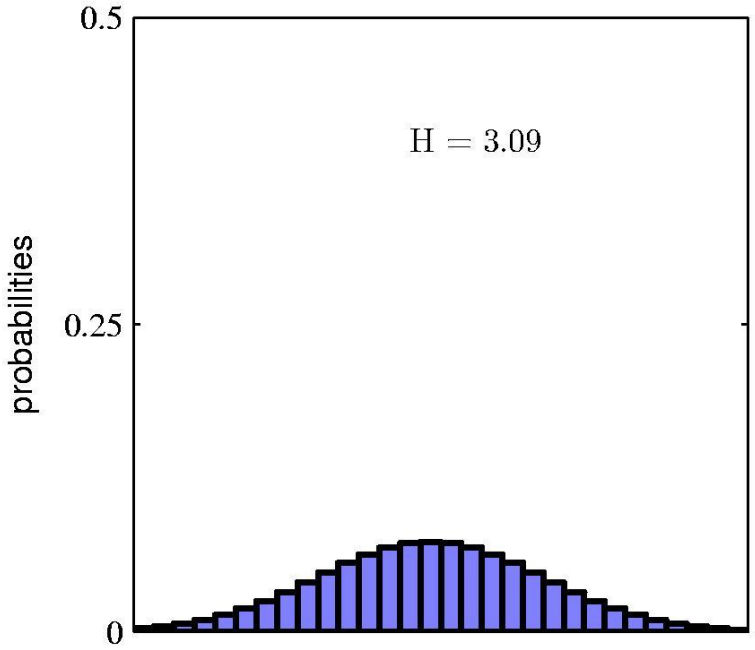
$$\mathrm{H}[x] = -\sum_{x} p(x) \log_2 p(x)$$

Important quantity in

- coding theory
- statistical physics
- machine learning

Entropy





The Kullback-Leibler Divergence

$$\begin{aligned}\mathrm{KL}(p\|q) &= -\int p(\mathbf{x})\ln q(\mathbf{x})\,\mathrm{d}\mathbf{x} - \left(-\int p(\mathbf{x})\ln p(\mathbf{x})\,\mathrm{d}\mathbf{x}\right) \\ &= -\int p(\mathbf{x})\ln\left\{\frac{q(\mathbf{x})}{p(\mathbf{x})}\right\}\mathrm{d}\mathbf{x}\end{aligned}$$

$$\mathrm{KL}(p\|q) \simeq \frac{1}{N}\sum_{n=1}^{N}\{-\ln q(\mathbf{x}_n|\boldsymbol{\theta}) + \ln p(\mathbf{x}_n)\}$$

$$\mathrm{KL}(p\|q) \geqslant 0 \qquad\qquad \mathrm{KL}(p\|q) \not\equiv \mathrm{KL}(q\|p)$$

Mutual Information

$$\begin{aligned} \mathrm{I}[\mathbf{x},\mathbf{y}] &\equiv \mathrm{KL}(p(\mathbf{x},\mathbf{y})\|p(\mathbf{x})p(\mathbf{y})) \\ &= -\iint p(\mathbf{x},\mathbf{y}) \ln\left(\frac{p(\mathbf{x})p(\mathbf{y})}{p(\mathbf{x},\mathbf{y})}\right) \mathrm{d}\mathbf{x}\,\mathrm{d}\mathbf{y} \end{aligned}$$

$$\mathrm{I}[\mathbf{x},\mathbf{y}] = \mathrm{H}[\mathbf{x}] - \mathrm{H}[\mathbf{x}|\mathbf{y}] = \mathrm{H}[\mathbf{y}] - \mathrm{H}[\mathbf{y}|\mathbf{x}]$$

Likelihood / Prior / Posterior

- A hypothesis (model, function, parameter set, weights) is denoted as *h*; it is one member of the hypothesis space *H*
- A set of training examples is denoted as *D*, a collection of (**x**, y) pairs for training
- Pr(*h*) – the *prior probability of the hypothesis* – without observing any training data, what is the probability that *h* is the target function we want?

Adapted from Rebecca Hwa

Likelihood / Prior / Posterior

- Pr(D) – the *prior probability of the observed data* – chance of getting the particular set of training examples D
- Pr($h|D$) – the *posterior probability of h* – what is the probability that h is the target given that we have observed D?
- Pr($D|h$) – the probability of getting D if h were true (a.k.a. *likelihood of the data*)
- Pr($h|D$) = Pr($D|h$)Pr(h)/Pr(D)

MLE and MAP Estimation

Maximum likelihood estimation (MLE):

$h_{ML} = \text{argmax } Pr(D|h)$

Maximum-a-posteriori (MAP) estimation:

$h_{MAP} = \text{argmax}_h \, Pr(h|D)$

$= \text{argmax}_h \, Pr(D|h)Pr(h)/Pr(D)$

$= \text{argmax}_h \, Pr(D|h)Pr(h)$

Classification via regression

- Suppose we ignore the fact that the target output y is binary (e.g., $0/1$) rather than a continuous variable
- So we will estimate a linear regression function

$$\begin{aligned} f(\mathbf{x}; \mathbf{w}) &= w_0 + w_1 x_1 + \ldots + w_d x_d \\ &= w_0 + \mathbf{x}^T \mathbf{w}_1, \end{aligned}$$

  based on the available data as before.

- Objective we want to minimize:

$$J_n(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i; \mathbf{w}))^2$$

Classification via regression cont'd

- We can use the resulting regression function

$$f(\mathbf{x}; \hat{\mathbf{w}}) = w_0 + \mathbf{x}^T \hat{\mathbf{w}}_1,$$

  to classify any new (test) example $\mathbf{x}$ according to

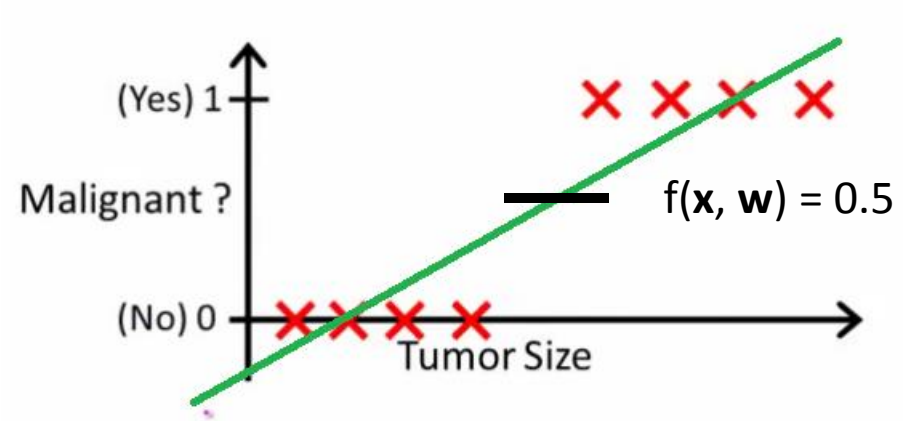
  label $= 1$ if $f(\mathbf{x}; \mathbf{w}) > 0.5$, and label $= 0$ otherwise

- $f(\mathbf{x}; \hat{\mathbf{w}}) = 0.5$ therefore defines a linear *decision boundary* that partitions the input space into two class specific regions (half spaces)

Classification via regression cont'd

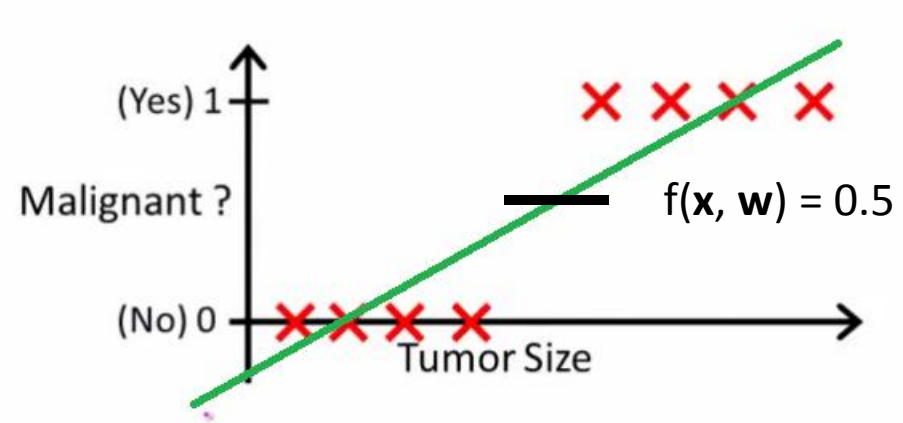
- Given the dissociation between the objective (classification) and the estimation criterion (regression) it is not clear that this approach leads to sensible results

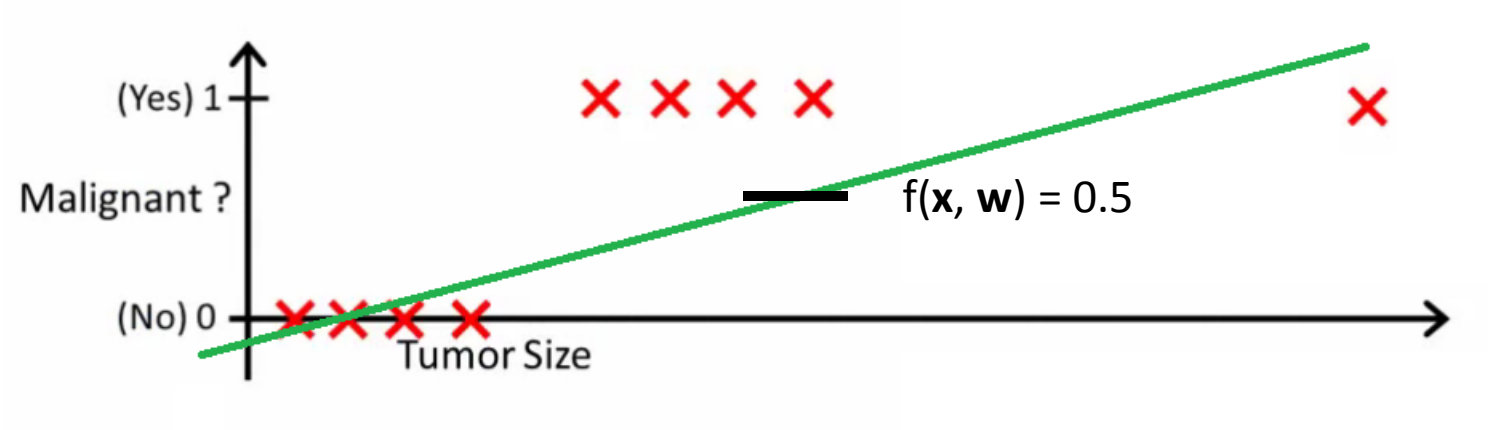
Classification via regression cont'd



Figures adapted from from Andrew Ng

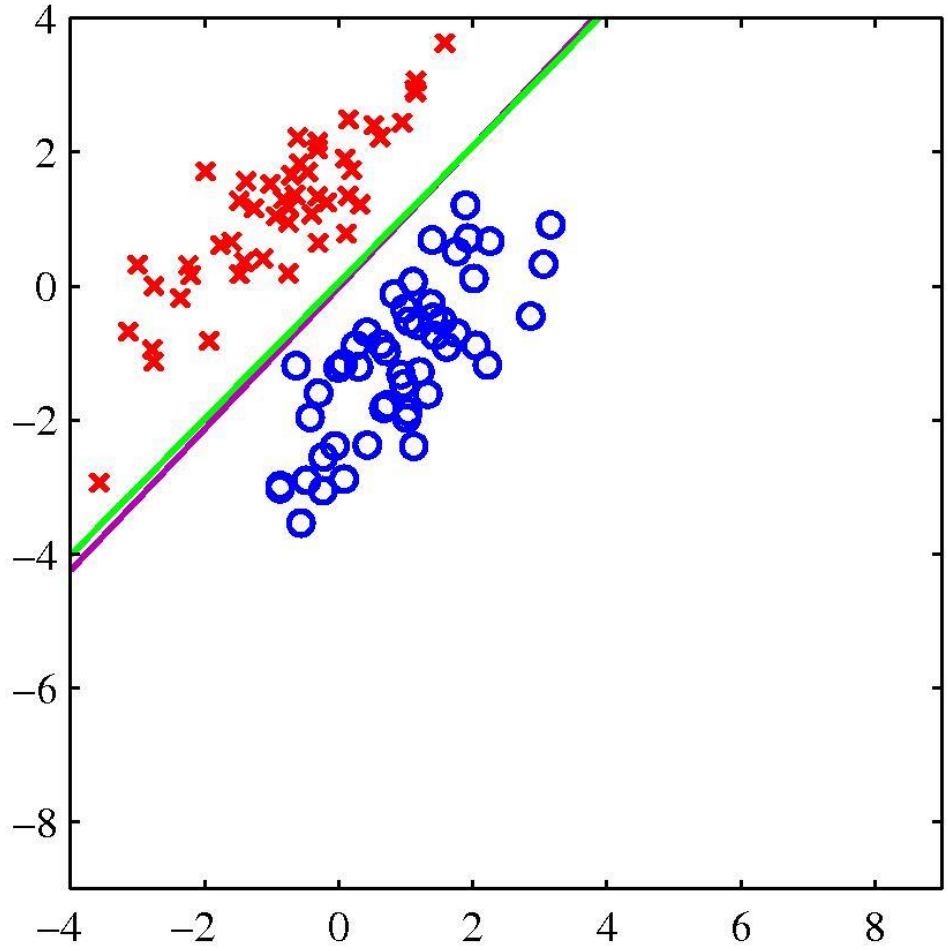
Classification via regression cont'd

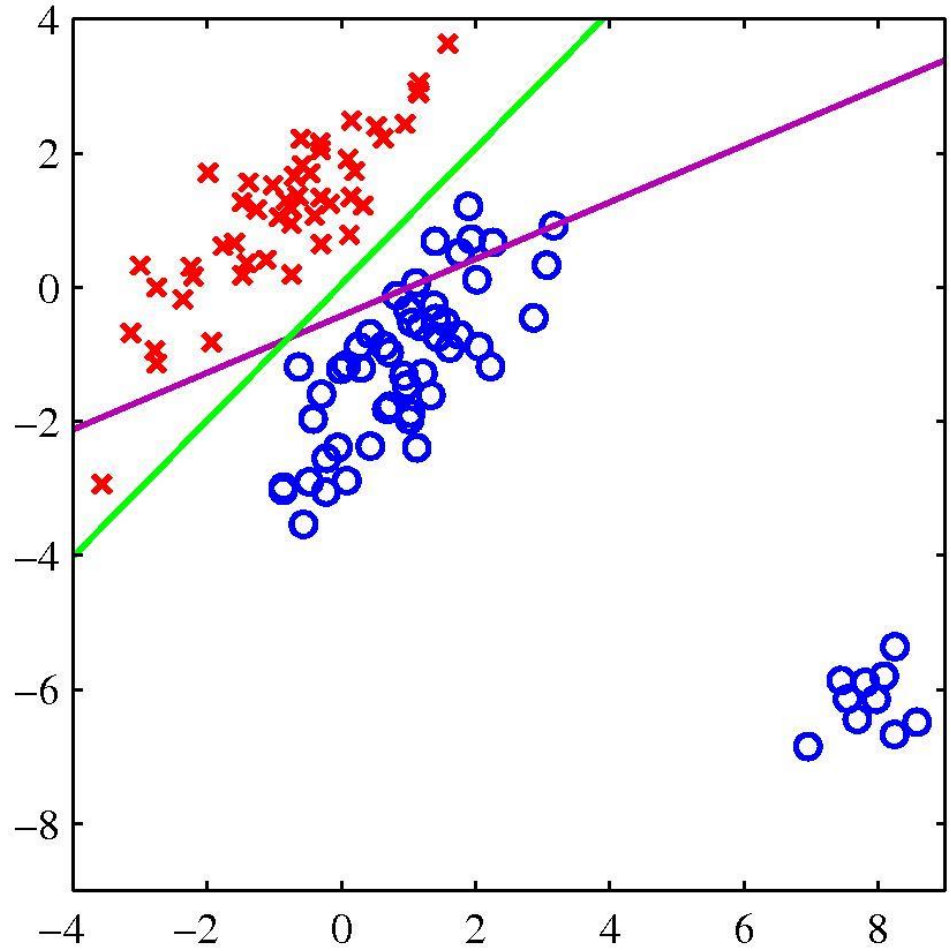




Figures adapted from from Andrew Ng

The effect of outliers: Another example





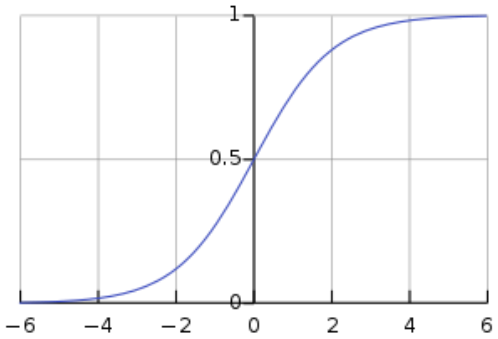
Magenta = least squares, green = logistic regression

Figures from Bishop

Logistic regression

- Also has “regression” in the name but it’s a method for classification
- Also uses a linear combination of the features to predict label, but in a slightly different way
- Fit a *sigmoid* function to model the *probability* of the data belonging to a certain class

P(y=1|x)

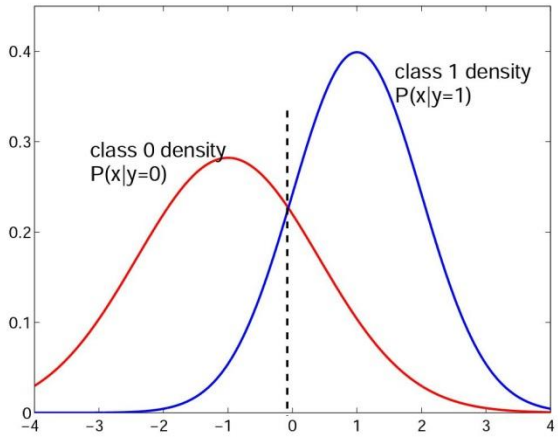


f(x) = dot(w, x) + b

Background: simple decision theory

- Suppose we know the class-conditional densities $p(\mathbf{x}|y)$ for $y = 0, 1$ as well as the overall class frequencies $P(y)$.

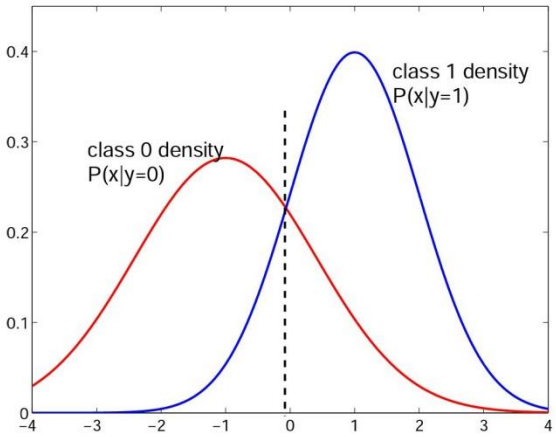
  How do we decide which class a new example $\mathbf{x}'$ belongs to so as to minimize the overall probability of error?



Background: simple decision theory

- Suppose we know the class-conditional densities $p(\mathbf{x}|y)$ for $y = 0, 1$ as well as the overall class frequencies $P(y)$.

  How do we decide which class a new example $\mathbf{x}'$ belongs to so as to minimize the overall probability of error?



The minimum probability of error decisions are given by

$$y' = \arg\max_{y=0,1} \{ p(\mathbf{x}'|y)P(y) \}$$

$$= \arg\max_{y=0,1} \{ P(y|\mathbf{x}') \}$$

Logistic regression

- The optimal decisions are based on the posterior class probabilities $P(y|\mathbf{x})$. For binary classification problems, we can write these decisions as

$$y = 1 \text{ if } \quad \log \frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})} > 0$$

  and $y = 0$ otherwise.

Logistic regression

- The optimal decisions are based on the posterior class probabilities $P(y|\mathbf{x})$. For binary classification problems, we can write these decisions as

$$y = 1 \text{ if } \quad \log \frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})} > 0$$

  and $y = 0$ otherwise.

- We generally don't know $P(y|\mathbf{x})$ but we can parameterize the possible decisions according to

$$\log \frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})} = f(\mathbf{x}; \mathbf{w}) = w_0 + \mathbf{x}^T \mathbf{w}_1$$

Logistic regression cont'd

- Our log-odds model

$$\log \frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})} = w_0 + \mathbf{x}^T \mathbf{w}_1$$

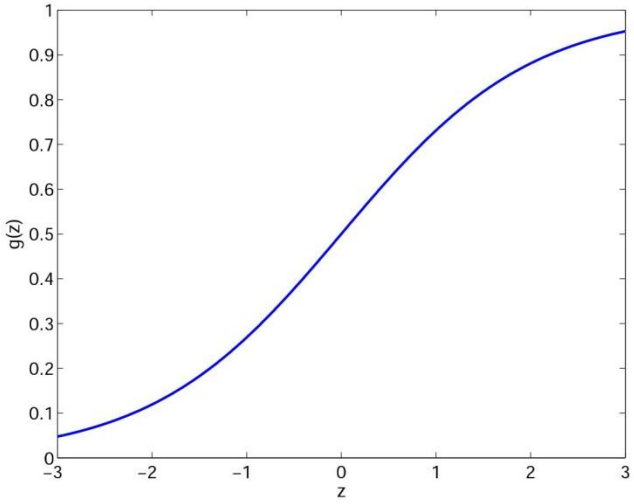
gives rise to a specific form for the conditional probability over the labels (the logistic model):

$$P(y=1|\mathbf{x}, \mathbf{w}) = \sigma\left(w_0 + \mathbf{x}^T \mathbf{w}_1\right)$$

where

$$\sigma(z) = (1 + \exp(-z))^{-1}$$

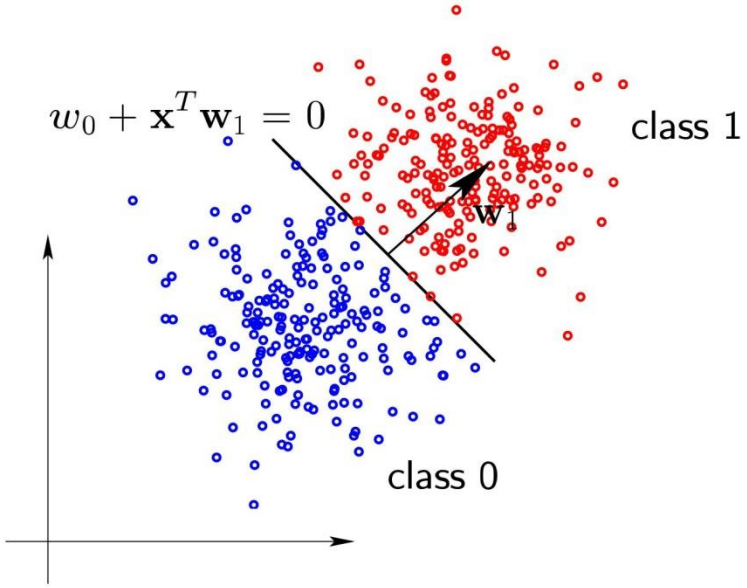
is a logistic "squashing function" that turns linear predictions into probabilities



Logistic regression: decisions

- Logistic regression models imply a linear decision boundary

$$\log \frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})} = w_0 + \mathbf{x}^T \mathbf{w}_1 = 0$$



Fitting logistic regression models

- We can fit the logistic models using the maximum (conditional) log-likelihood criterion

$$l(D; \mathbf{w}) = \sum_{i=1}^{n} \log P(y_i | \mathbf{x}_i, \mathbf{w})$$

where

$$P(y = 1 | \mathbf{x}, \mathbf{w}) = \sigma \left(w_0 + \mathbf{x}^T \mathbf{w}_1 \right)$$

- Solution: find roots of $\underbrace{\left(y_i - P(y_i = 1 | \mathbf{x}_i, \mathbf{w})\right)}_{\text{prediction error}} \begin{bmatrix} 1 \\ \mathbf{x}_i \end{bmatrix} = 0$

Stochastic gradient ascent

- We can try to maximize the log-likelihood in an *on-line* or incremental fashion.

  Given each training input $\mathbf{x}_i$ and the binary (0/1) label y_i, we can change the parameters $\mathbf{w}$ slightly to increase the corresponding log-probability

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} + \eta \frac{\partial}{\partial \mathbf{w}} \log P(y_i | \mathbf{x}_i, \mathbf{w}) \\ &= \mathbf{w} + \eta \underbrace{\left(y_i - P(y_i = 1 | \mathbf{x}_i, \mathbf{w})\right)}_{\text{prediction error}} \begin{bmatrix} 1 \\ \mathbf{x}_i \end{bmatrix} \end{aligned}$$

  where η is the *learning rate*.

Whiteboard: solution

Logistic Regression / MLE Example

- Want to find the weight vector that gives us the highest P(y_i|x_i, w),
- where P(y_i=1|x_i, w) = 1 / (1 + exp(-w'*x))
- Consider two weight vectors and three samples, with corresponding likelihoods:

	P(y_1 = 1 | x_1, w_i)	P(y_2 = 1 | x_2, w_i)	P(y_3 = 1 | x_3, w_i)
w_1	0.3	0.1	0.4
w_2	0.7	0.8	0.2
True label:	*1*	*0*	*1*

Logistic Regression / MLE Example

- Then the value of the objective for w_i is:
  P(y_1 = 1 | x_1, w_i) *
  (1 – P(y_2 = 1 | x_2, w_i)) *
  P(y_3 = 1 | x_3, w_i)
- So the score for w_1 is: 0.3 * 0.9 * 0.4
- And the score for w_2 is: 0.7 * 0.2 * 0.2
- Thus, w_1 is a better weight vector = model

Plan for this lecture

- The simplest classifier: K-Nearest Neighbors
  - Algorithm and example use
  - Generalizing: Distance metrics, weighing neighbors
  - Problems: curse of dimensionality, picking K
- Logistic regression
  - Probability: review
  - Linear regression for classification?
  - Maximum likelihood solution for *logistic* regression
  - Related algorithm: perceptron

The perceptron algorithm

- Rosenblatt (1962)
- Prediction rule: $y(\mathbf{x}) = f\left(\mathbf{w}^{\mathrm{T}}\overbrace{\boldsymbol{\phi}(\mathbf{x})}^{\mathbf{x}}\right)$

  where $$f(a) = \begin{cases} +1, & a \geqslant 0 \\ -1, & a < 0 \end{cases}$$

- Want: $\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)t_n > 0$ (t_n = +1 or -1)
- Loss: $E_{\mathrm{P}}(\mathbf{w}) = -\sum_{n\in\mathcal{M}} \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}_n t_n$

  (just using the Misclassified examples)

The perceptron algorithm

- Loss: $$E_{\mathrm{P}}(\mathbf{w}) = -\sum_{n \in \mathcal{M}} \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}_n t_n$$

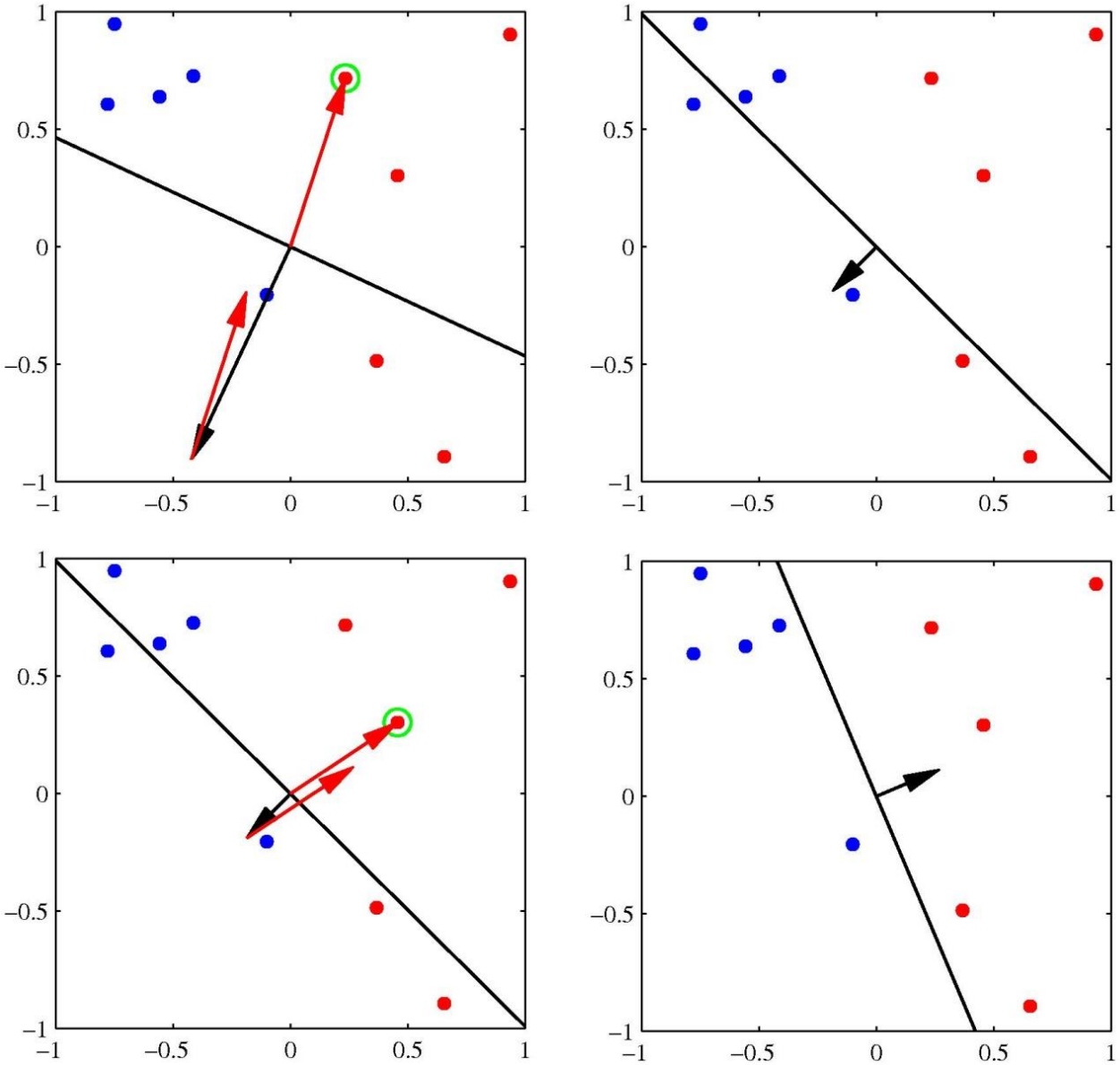
- Learning algorithm update rule:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_{\mathrm{P}}(\mathbf{w}) = \mathbf{w}^{(\tau)} + \eta \boldsymbol{\phi}_n t_n$$

- Interpretation:
  - If sample is misclassified and is positive, make the weight vector more like it
  - If sample is misclassified and negative… *unlike* it

The perceptron algorithm (red=pos)

w = w + x
(x is pos)



Figures from Bishop

Summary: Tradeoffs of classification methods thus far

- Nearest neighbors
  - Non-parametric method; basic formulation cannot ignore/focus on different feature dimensions
  - Slow at test time (large search problem to find neighbors)
  - Need to store all data points (unlike SVM, coming next)
  - Decision boundary not necessarily linear
  - Naturally handles multiple classes
- Logistic regression (a *classification* method)
  - Models the *probability* of a label given the data
  - Decision boundary corresponds to $\mathbf{w}^T \mathbf{x} = 0$ (a line)
- Perceptron
  - Same decision boundary as logistic regression (a line)
  - Simple update rule
  - Won't converge for non-linearly-separable data