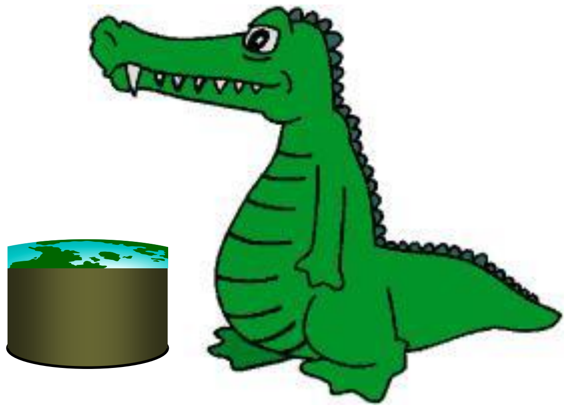
CAP4770/5771 Introduction to Data Science Fall 2016

University of Florida, CISE Department
Prof. Daisy Zhe Wang



Based on notes from CS194 at UC Berkeley by Michael Franklin, John Canny, and Jeff Hammerbacher

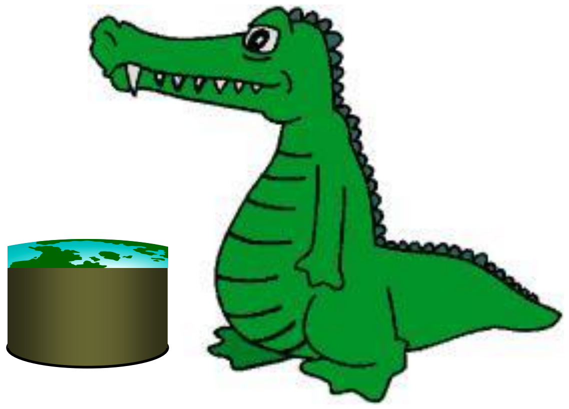
Logistics

- Lab 3 – deadline yesterday
- Lab 4 – coming Monday
- Midterm Dates voting
  - conducted in class
  - 10/10 (Monday) → 21 vote against
  - 10/12 (Wednesday) → 9 vote against
  - Midterm will be held on 10/12
  - Coverage: up to lecture 7

Review

- Tabular data manipulation
  - Single table operations
  - Join, Aggregation
  - Data cube operations
- Similarity Measures
  - Editing distance – fuzzy join
- Quality Metrics
  - Precision vs. recall curve
  - true positive vs. false positive curve

Supervised Learning: Regression & Classification



Outline

- Machine Learning
  - Supervised
  - Unsupervised
- Supervised Learning
  - Classification and Regression
  - K-NN
  - Naïve Bayes
  - Linear/Logistic Regression
  - SVM
  - Decision Tree and Random Forest

Machine Learning

- **Supervised:** We are given input samples (X) and output samples (y) of a function y = f(X). We would like to "learn" f, and evaluate it on new data. Types:
  - **Classification:** y is discrete (class labels).
  - **Regression:** y is continuous, e.g. linear regression.

- **Unsupervised:** Given only samples X of the data, we compute a function f such that y = f(X) is "simpler".
  - **Clustering:** y is discrete
  - Y is continuous: e.g., **Matrix factorization**

Machine Learning

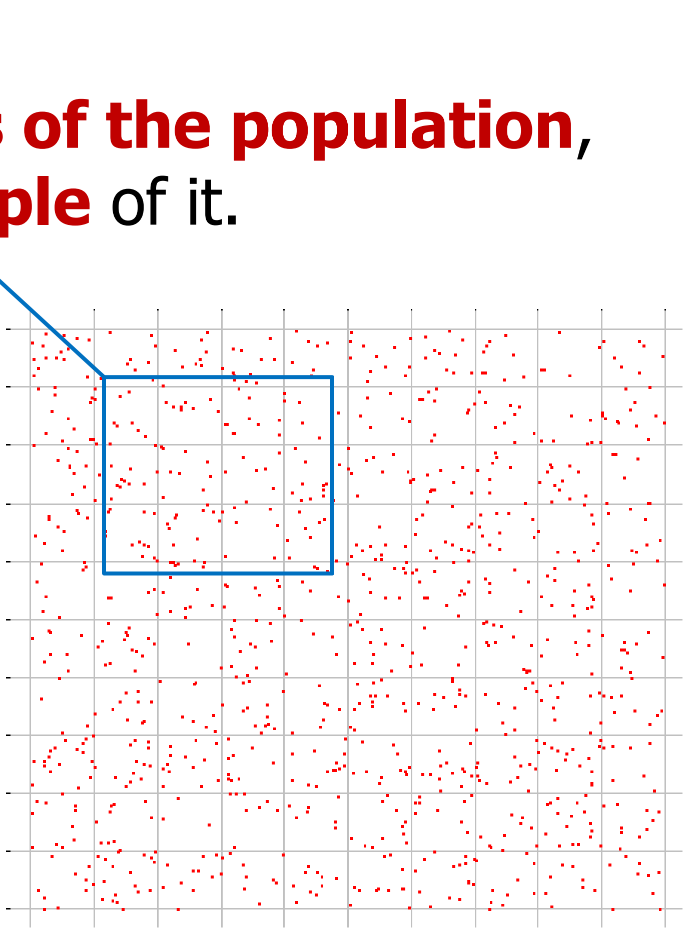
- **Supervised:**
  - Is this image a cat, dog, car, house?
  - How would this user score that restaurant?
  - Is this email spam?
  - Is this blob a supernova?
- **Unsupervised**
  - Cluster some hand-written digit data into 10 classes.
  - What are the top 20 topics in Twitter right now?

Techniques

- **Supervised Learning:**
  - kNN (k Nearest Neighbors)
  - Naïve Bayes
  - Linear + Logistic Regression
  - Support Vector Machines
  - Decision Tree
  - Random Forests
  - Neural Networks
- **Unsupervised Learning:**
  - K-means Clustering
  - Topic Models
  - Neural Networks

Predicting from Samples

- Most datasets are **samples** from an **infinite population**.
- We are most interested in **models of the population**, but we have access only to a **sample** of it.



For datasets consisting of (X,y)

- features X + label y

a model is a prediction y = f(X)

We train on a training sample D and we denote the model as $f_D(X)$

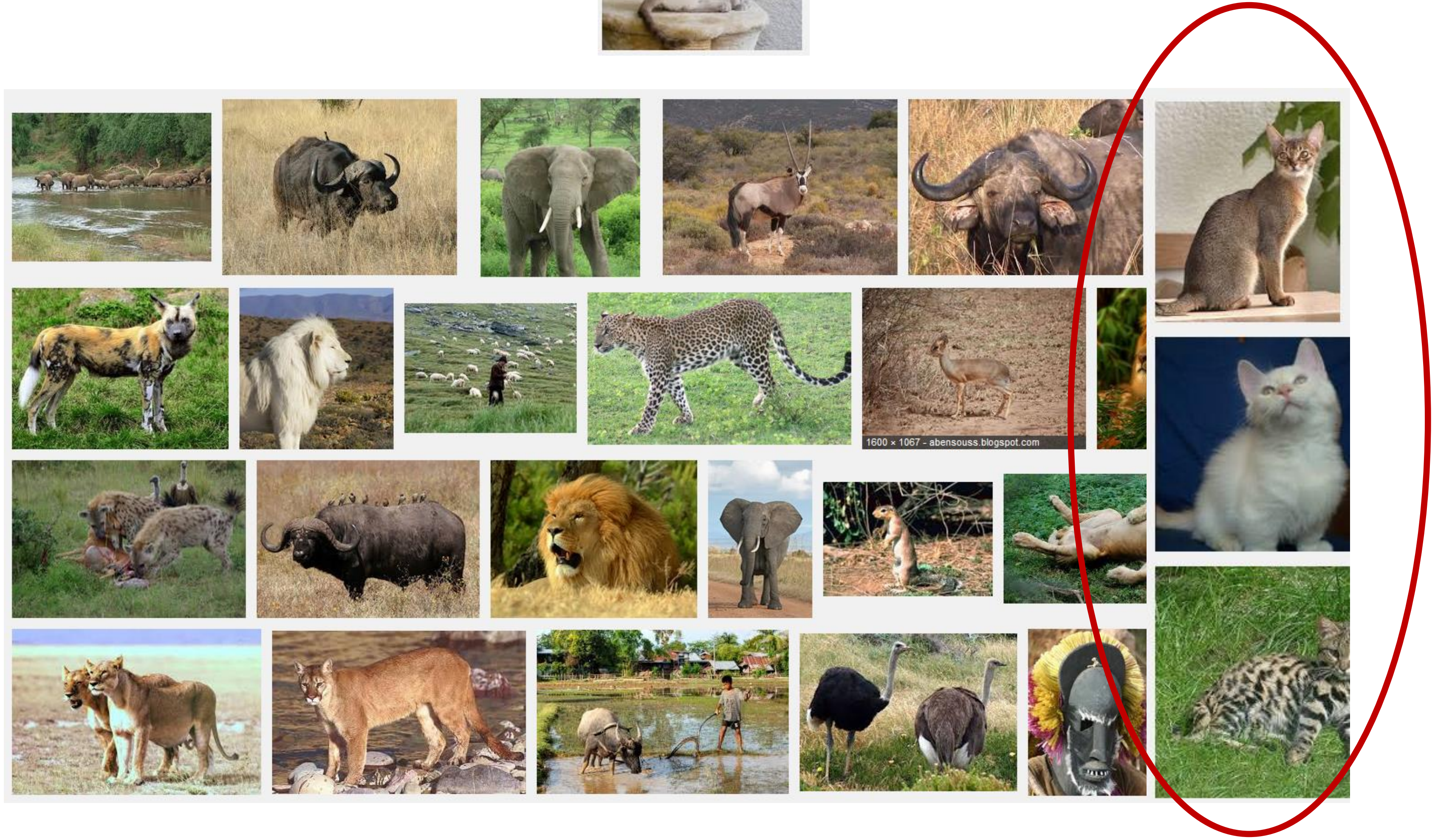
k-Nearest Neighbors

- Given a query item:
- Find k closest matches in a labeled dataset ↓

Return the most frequent Label

k-Nearest Neighbors

k = 3 votes for “cat”

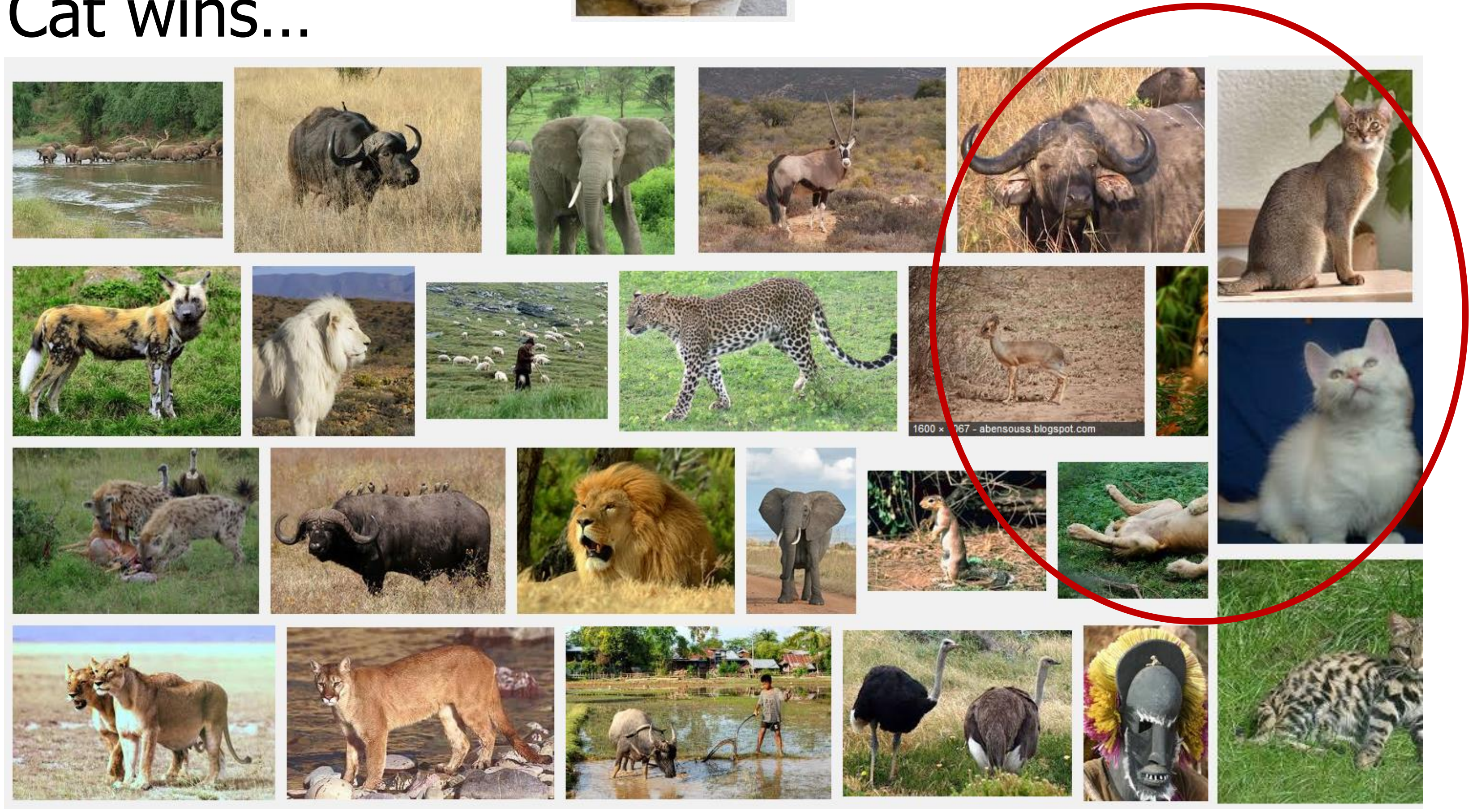


k-Nearest Neighbors

2 votes for cat,

1 each for Buffalo,

Cat wins…



k-NN Flavors

Classification:

- Model is y = f(X), y is from a discrete set (labels).
- Given X, compute y = majority vote of the k nearest neighbors.
- Can also use a weighted vote* of the neighbors.

Regression:

- Model is y = f(X), y is a real value.
- Given X, compute y = average value of the k nearest neighbors.
- Can also use a weighted average* of the neighbors.

- Weight function is usually the inverse distance. Why?

k-NN issues

The Data is the Model

- No training needed.
- Accuracy generally improves with more data.
- Matching is simple and fairly fast if data fits in memory, but can be run off disk.

Minimal Configuration:

- Only parameter is k (number of neighbors)
- But two other choices are important:
  - Weighting of neighbors (e.g. inverse distance)
  - Similarity metric

K-NN distance measures

- **Euclidean Distance:** Simplest, fast to compute

$$d(x, y) = \|x - y\|$$

- **Cosine Distance:** Good for documents, images, etc.

$$d(x, y) = 1 - \frac{x \cdot y}{\|x\| \|y\|}$$

- **Jaccard Distance:** For set data:

$$d(X, Y) = 1 - \frac{|X \cap Y|}{|X \cup Y|}$$

K-NN distance measures II

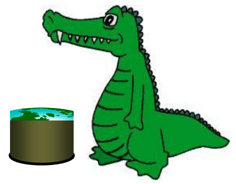
- **Manhattan Distance:** Coordinate-wise distance

$$d(x,y) = \sum_{i=1}^{n} |x_i - y_i|$$

- **Edit Distance:** for strings, especially genetic data.

- **Hamming Distance:** For string data:

$$d(x,y) = \sum_{i=1}^{n} (x_i \neq y_i)$$



Some Similarity Measures

Handle Typographical errors

- Equality on a boolean predicate
- Edit distance
  - Levenstein, Smith-Waterman, Affine
- Set similarity
  - Jaccard, Dice
- Vector Based
  - Cosine similarity, TFIDF

Good for Text like reviews/ tweets

Good for Names

- Alignment-based or Two-tiered
  - Jaro-Winkler, Soft-TFIDF, Monge-Elkan
- Phonetic Similarity
  - Soundex
- Translation-based
- Numeric distance between values
- Domain-specific

Useful for abbreviations, alternate names.

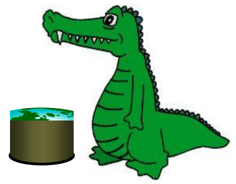
From: Getoor & Machanavajjhala: “Entity Resolution Tutorial”, VLDB 2012

kNN and the curse of dimensionality

The curse of dimensionality refers to phenomena that occur in high dimensions (100s to millions) that do not occur in low-dimensional (e.g. 3-dimensional) space.

In particular data in high dimensions are much sparser (less dense) than data in low dimensions.

For kNN, that means there are less points that are very close in feature space (very similar), to the point we want to predict.



kNN and the curse of dimensionality

Example: Consider a collection of uniformly random points in the unit cube. In one dimension, the average squared Euclidean distance between any two points is:

$$\int_0^1 \int_0^1 (x-y)^2 dx\, dy = \frac{1}{6}$$

In N dimensions, we add up the squared differences for all N coordinates (because the coordinates are independent in a uniform random cube), giving:

$$d^2 = \mathrm{E}[||x-y||^2] = \frac{N}{6}$$

So the euclidean distance scales as $\sqrt{N}$

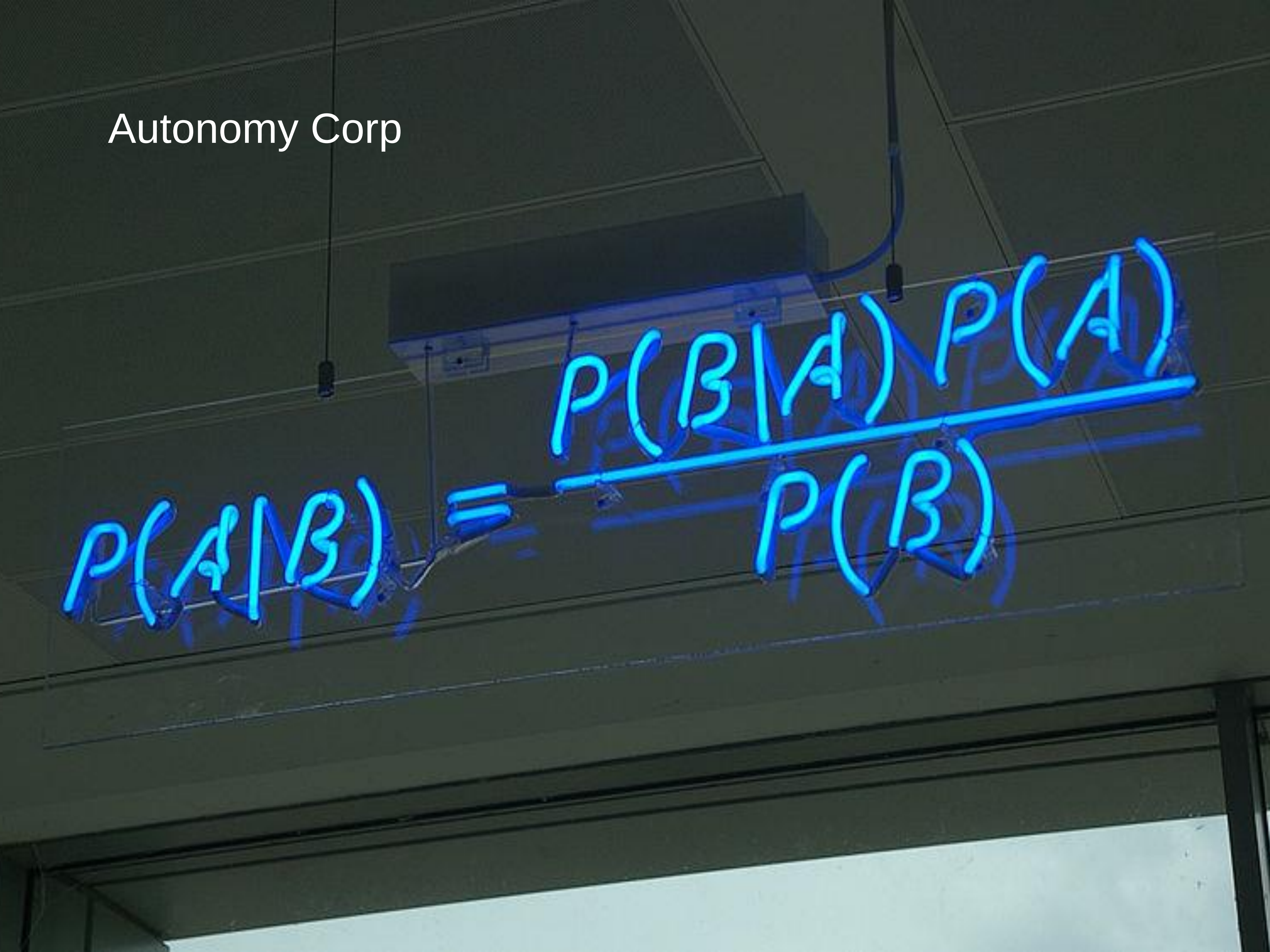
kNN and the curse of dimensionality

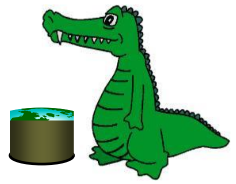
From this perspective, its surprising that kNN works at all in high dimensions.

Luckily real data are not like random points in a high-dimensional cube. Instead they live in **dense clusters** and near **much lower-dimensional surfaces**.

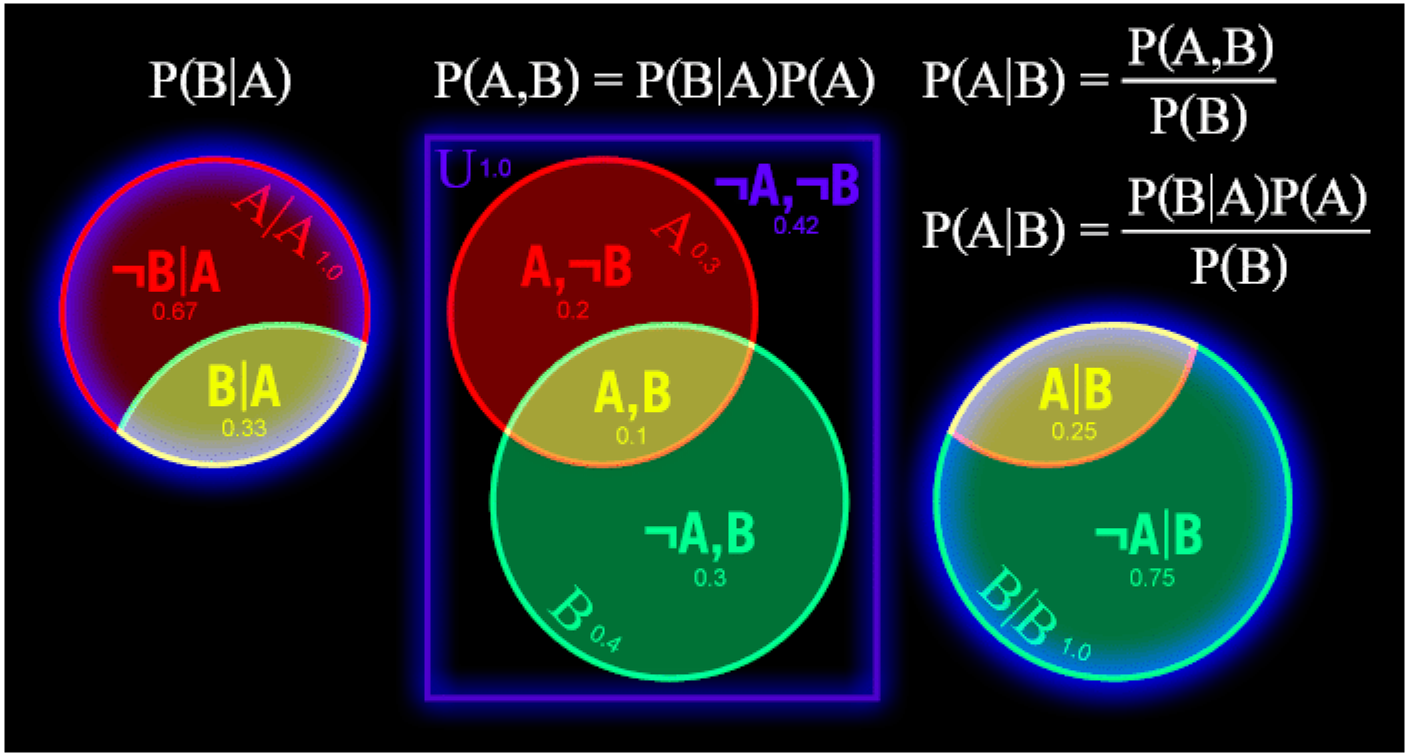
Finally, points can be very "similar" even if their euclidean distance is large. E.g. documents with the same few dominant words (with tf-idf weighing) are likely to be on the same topic.

Autonomy Corp





Bayes' Theorem

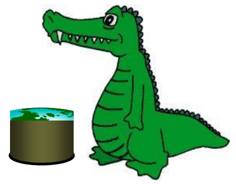


Assumes probability of event is proportional to area of set.

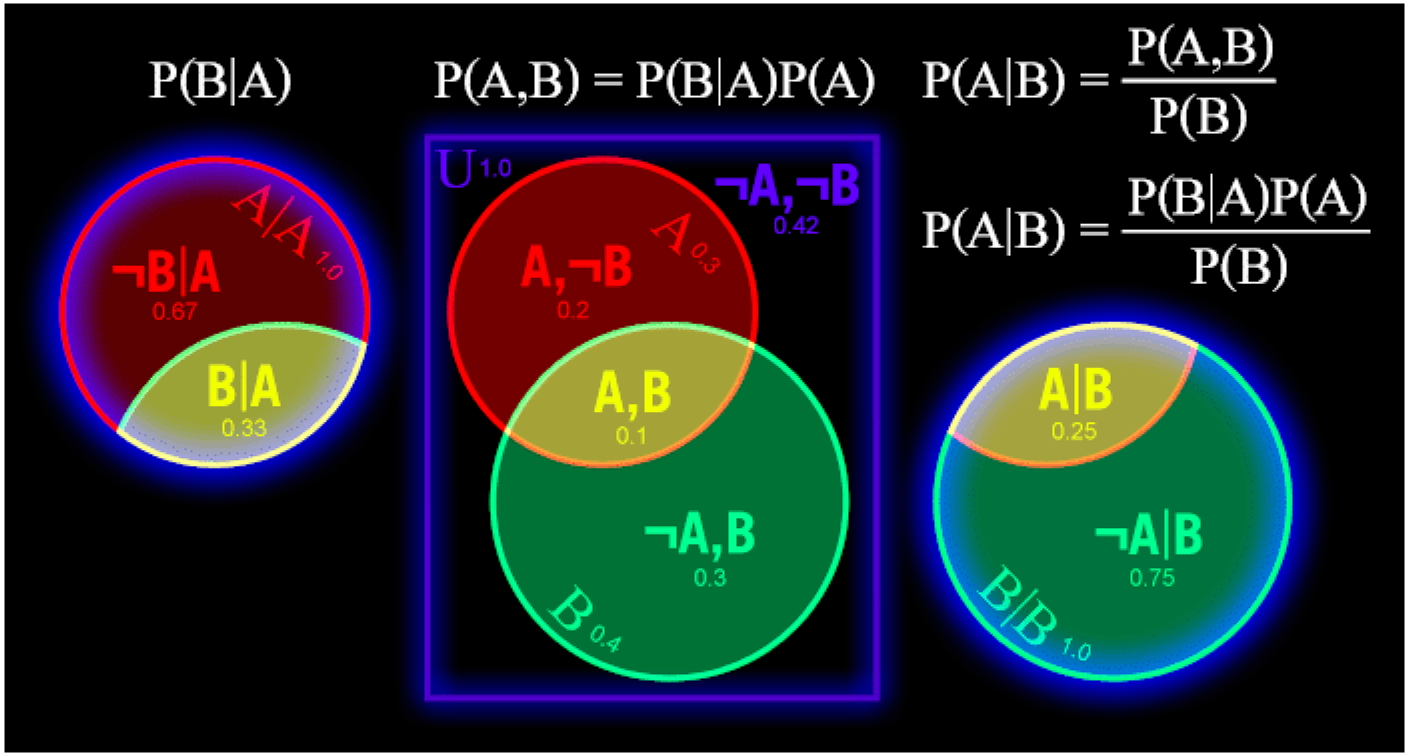
P(A) = area of A.

P(A,B) = area of A intersect B.

P(A|B) = P(A,B) / P(B)



Bayes' Theorem



The theorem follows by writing:

$$P(A|B)P(B) \;=\; P(A,B) \;=\; P(B|A)P(A)$$

From which we get:

$$P(A|B) = P(B|A)\,P(A) \,/\, P(B)$$

Bayes' Theorem

P(A|B) = probability of A given that B is true.

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

In practice we are most interested in dealing with events e and data D.

e = "I have a cold"
D = "runny nose," "watery eyes," "coughing"

$$P(e|D) = \frac{P(D|e)P(e)}{P(D)}$$

So Bayes' theorem is "diagnostic".

Bayes’ Terminology

D = Data, e = some event

$$P(e|D) = \frac{P(D|e)P(e)}{P(D)}$$

P(e) is called the **prior probability** of e. Its what we know (or think we know) about e with no other evidence.

P(D|e) is the **conditional probability** of D given that e happened, or just the likelihood of D. This can often be measured or computed precisely.

P(e|D) is the **posterior probability** of e given D. It’s the answer we want, or the way we choose a best answer.

You can see that the posterior is heavily colored by the prior, so Bayes’ has a GIGO liability. e.g. its not used to test hypotheses.

Naïve Bayes Classifier

Let’s assume we have an instance (e.g. a document d) with a set of features $(X_1, \ldots, X_k)$ and a set of classes $\{c_j\}$ to which the document might belong.

We want to find the **most likely class** that the document belongs to, given its features.

The joint probability of the class and features is:

$$\Pr(X_1, \ldots, X_k, c_j)$$

Naïve Bayes Classifier

Key Assumption: (Naïve) the features are **generated independently** given c_j. Then the joint probability factors:

$$\Pr(X, c_j) = \Pr(X_1, \dots, X_k \mid c_j) \Pr(c_j) = \Pr(c_j) \prod_{i=1}^{k} \Pr(X_i | c_j)$$

We would like to figure out the **most likely class for** (i.e. to classify) the document, which is the c_j which maximizes:

$$\Pr(c_j \mid X_1, \dots, X_k)$$

Naïve Bayes Classifier

Now from Bayes we know that:

$$\Pr\left(\overset{\textbf{A}}{c_j} \mid \overset{\textbf{B}}{X_1, \ldots, X_k}\right) = \Pr\left(\overset{\textbf{B}}{X_1, \ldots, X_k} \mid \overset{\textbf{A}}{c_j}\right) \overset{\textbf{A}}{\Pr\left(c_j\right)} \,/\, \Pr(\overset{\textbf{B}}{X_1, \ldots, X_k})$$

But to choose the best c_j, we can ignore $\Pr(X_1, \ldots, X_k)$ since it’s the same for every class. So we just have to maximize:

$$\Pr\left(X_1, \ldots, X_k \mid c_j\right) \Pr\left(c_j\right)$$

So finally we pick the category c_j that maximizes:

$$\Pr\left(X_1, \ldots, X_k \mid c_j\right) \Pr\left(c_j\right) = \Pr\left(c_j\right) \prod_{i=1}^{k} \Pr\left(X_i | c_j\right)$$

Data for Naïve Bayes

In order to find the best class, we need two pieces of data:

- $\Pr(c_j)$ the prior probability for the class c_j.
- $\Pr(X_i|c_j)$ the conditional probability of the feature X_i given the class c_j.

Data for Naïve Bayes

For these two data, we only need to record counts:

- $\Pr(c_j) = \frac{N_d(c_j)}{N_d}$
- $\Pr(X_i | c_j) = \frac{N_w(X_i, c_j)}{N_w(c_j)}$

Where $N_d(c_j)$ is the number of documents in class c_j, N_d is the total number of documents.

$N_w(X_i, c_j)$ is the number of times X_i occurs in a document in c_j, and and $N_w(c_j)$ is the total number of features in all docs in c_j.

“Training” Naïve Bayes

So there is no need to train a Naïve Bayes classifier, only to accumulate the N_w and N_d counts.

But count data is only an approximation to those probabilities

and a count of zero is problematic (why)?

Naïve Bayes and Smoothing

But count data is only an approximation to those probabilities however, and a count of zero is problematic: It forces the classifier not to choose classes with zero counts.

Instead of direct count ratios, we can use **Laplace Smoothing**:

$$p = \frac{N_1 + \alpha}{N_2 + \beta}$$

With constants α and β. These values push p toward a **prior** of α/β.

These constants can either be set based on prior knowledge, or learned during a training phase.

Practical Naïve Bayes

We want the category c_j that maximizes:

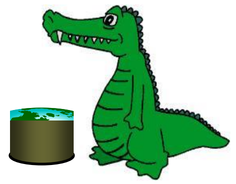
$$\Pr(X_1, \dots, X_k \mid c_j) \Pr(c_j) = \Pr(c_j) \prod_{i=1}^{k} \Pr(X_i | c_j)$$

The log of this expression has the same maximum:

$$\log \Pr(c_j) + \sum_{i=1}^{k} \log \Pr(X_i | c_j)$$

This formula is much better numerically (avoids floating point underflow) and involves simple vector operations.

Complexity of evaluating this expression is proportional to k, the number of features.



Good, Bad and Ugly of NB Classifiers

- **Simple and fast.** Depend only on term frequency data for the classes. One pass over the training dataset, no iteration.
- **Compact model.** Size is proportional to numfeats x numclasses.
- **Very well-behaved numerically.** Term weight depends only on frequency of that term. Decoupled from other terms.
- **Can work very well with sparse data**, where combinations of dependent terms are rare.
- **Subject to error and bias** when term probabilities are not independent.
- **Can’t model patterns** in the data.
- **Typically not as accurate** as other methods for the above reasons.