Features and Kernels

COS 424/524, SML 302: Fundamentals of Machine Learning

Professor Engelhardt

COS 424/524, SML 302

Lecture 5

Features and kernels

In the previous lecture, we saw:

- Classification as a supervised learning problem;
- A simple generative model for classification: Naive Bayes model.

Today we will discuss:

- What are *features* of observed data? What are *useful* features?
- Alternative to feature definition: *kernel functions*.

Features

The word *feature* comes from Latin (factūra) meaning *a making*, and having the same root as the word *fact* (source: `dictionary.com`)

Feature: "a prominent or conspicuous part or characteristic"

In machine learning, a *feature* is "an individual measurable heuristic property of a phenomenon being observed." (source: `wikipedia`)

Key point: Engineering appropriate features to characterize a sample is essential to the performance of any analysis.

Indeed, good results from an analysis are often possible only through creative feature engineering.

Feature as a measurable property

Measurable, or quantifiable, property:

- counts (e.g., specific words, citations, views, alleles)
- rankings or orderings (e.g., Netflix ratings, sports, chess)
- quantification of a continuous value (e.g., height, RGB color, gene expression levels, time)
- proportions (e.g., document topics, ancestral populations in a genome)

Multiple features for a single sample are often grouped into a *feature vector*; we can manipulate this feature vector using linear algebra.

Why create features? Interpretability.

- Features associate samples with a point in p space
- When we hand-engineer features, we can label those p dimensions (e.g., vocabulary word, height, movie rating, genome mutation)
- When we use probabilistic models, we estimate the relationship between features and labels, and among samples via features
- These relationships give us *interpretability* – how is each named feature involved in the important patterns we find in our data

Without named features and without quantified relationships between features and patterns, we do not have interpretability.

Features of specific data types

Let's discuss available features for specific data types:

- Text data
- Image data
- Network data

Text data: emails

Let's look at some emails:

Example

Good day!
We considered your resume to be very attractive and we thought the vacant position in our company could be interesting for you.
We cooperate with different countries and currently we have many clients in the world.
Part-time and full-time employment are both currently important.
We offer a flat wage from $1500 up to $7000 per month.
The job offers a good salary so, interested candidates please registration on the our site: www.thinksmoney.com
Attention! Accept applications only on this and next week.
Respectively submitted
Personnel department

Email spam classification

Example

Hey,
I just saw that there's a chess tournament this saturday 1/17 at Charter, maybe your son can play:
http://www.chesskidsny.com/tournaments/PCSChessProgram.htm
http://www.chesskidsny.com/charter/PCStour_register.php (payment/registration)
contact XXX@gmail.com, Miguel XXX
$40 on site at 3:15 pm, maybe less online.
I think I mentioned that Tom XXX started the chess club at Community Park.
His email is thomasXXX@gmail.com
I hope the NYGC meeting is great.

Bag-of-words representation

On the Origin of Species.

Introduction.

When on board H.M.S. 'Beagle,' as naturalist, I was much struck with certain facts in the distribution of the inhabitants of South America, and in the geological relations of the present to the past inhabitants of that continent. These facts seemed to me to throw some light on the origin of species—that mystery of mysteries, as it has been called by one of our greatest philosophers. On my return home, it occurred to me, in 1837, that something might perhaps be made out on this question by patiently accumulating and reflecting on all sorts of facts which could possibly have any bearing on it. After five years' work I allowed myself to speculate on the subject, and drew up some short notes; these I enlarged in 1844 into a sketch of the conclusions, which then seemed to me probable: from that period to the present day I have steadily

species	1662
origin	352
evolution	0
future	33
modification	179
animals	362
imagine	26
extinct	171
exist	259
natural	524
complex	62
generation	108
selection	380
neutral	2
mutation	8
probability	166

Bag-of-words representation

Bag-of-words representation is often modeled with a *multinomial* distribution. Why? Natural alternatives?

- One bag-of-words feature vector is <document word count> rolls of a die with <dictionary size> sides
- Choose the dictionary carefully
- Can add Dirichlet prior (Laplace smoothing) to avoid underflow with rare/new words

Bag-of-words representation

Bag-of-words representation of text eliminates:

- word order
- specific n-grams
- possibility of disambiguation

More sophisticated text features?

How can I improve upon bag-of-words? (depends on goal)

- Additional features of the document: author, length, references
- Punctuation, capitalization, italics, bold.
- Average: sentence length, word length, paragraph length.
- Follow links, references, citations.
- Latent "topic" representations (future class)
- Sentiment analysis

Image analysis

















We want to classify these images. What features would be useful?

Bag-of-pixels

We could create a bag-of-pixels vector for an image:

- characterize colors by clustering RGB values;
- count the number of pixels in a image in each RGB bin

Colors do not capture an image in the same way that words capture a document.

Convolution and feature detection

- *Filtering:* replace each pixel by the weighted combination of that pixel and neighbor pixels; weights are determined by a *filter*
- This function is referred to as *convolution*

Linear filter

A linear filter produces an identical image:

0	0	0
0	1	0
0	0	0

A blurring filter produces a blurred image:

1	1	1
1	1	1
1	1	1

Image filters

0	0	0
0	1	0
0	0	0

1	1	1
1	1	1
1	1	1

`http://softwarebydefault.com`

Detecting features

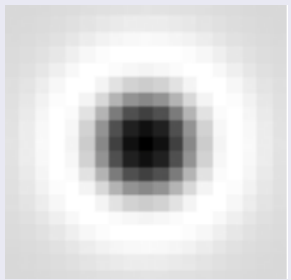
- We can use specific filters to identify features in an image
- Consider: edges, corners, *blobs*, patterns (e.g., polka-dots, stripes)

Linear filter

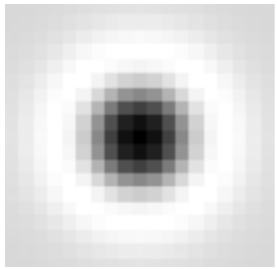
An edge filter finds edges

0	0	0
-1	1	0
0	0	0

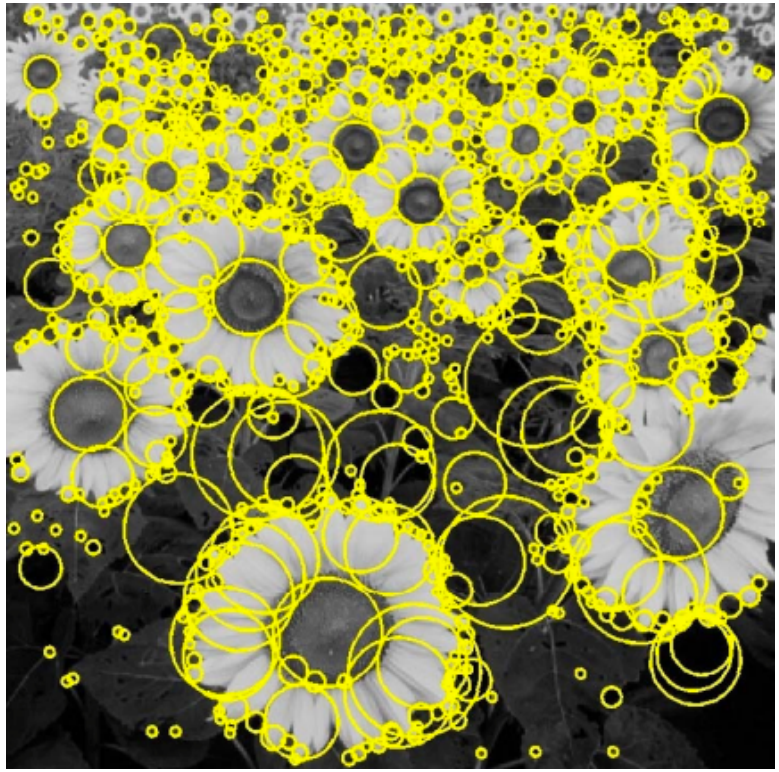
A blob feature finds *blobs*



Edge and blob detectors

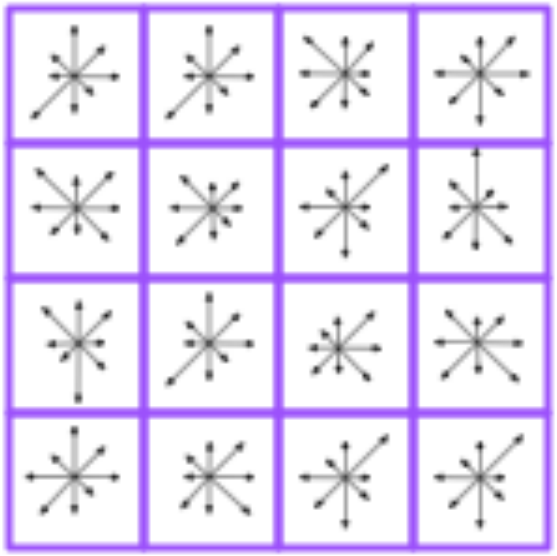


0	0	0
-1	1	0
0	0	0

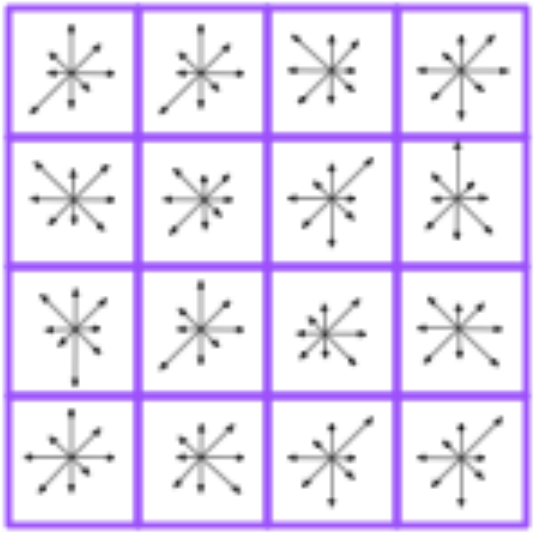


SIFT features

- *Scale-invariant feature transform (SIFT):* detects local features in an image [Lowe, 1999]
- Extract SIFT features: smooth and resample image; apply specific gradient filters (128 descriptors)
- SIFT features are invariant to scale, noise, and illumination

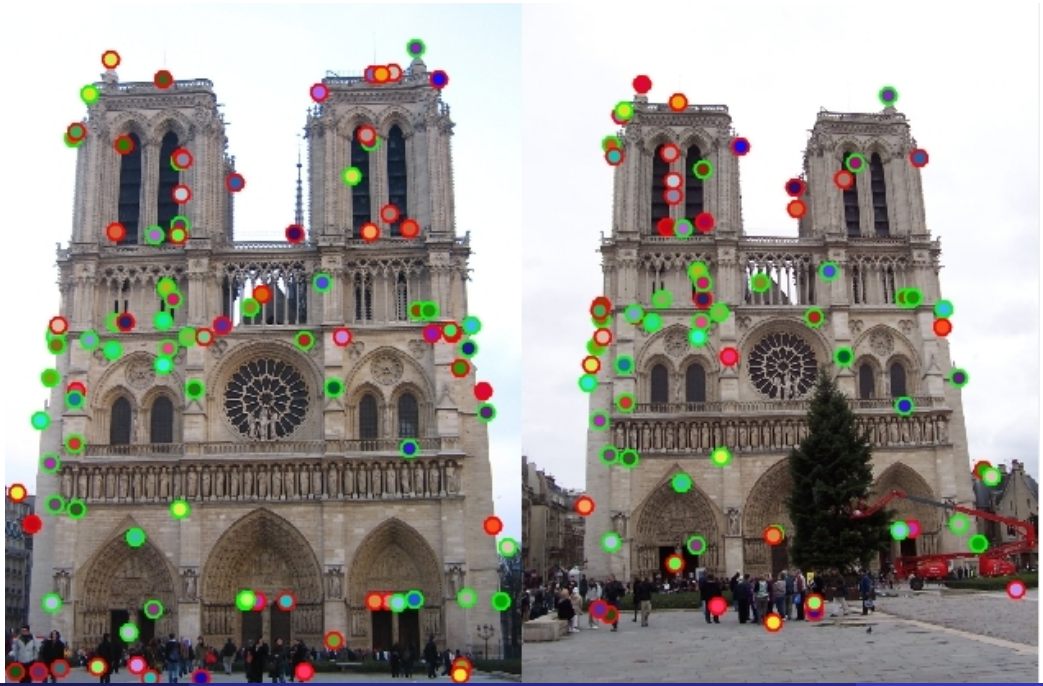


SIFT features (borrowed from Sharma, GA Tech)



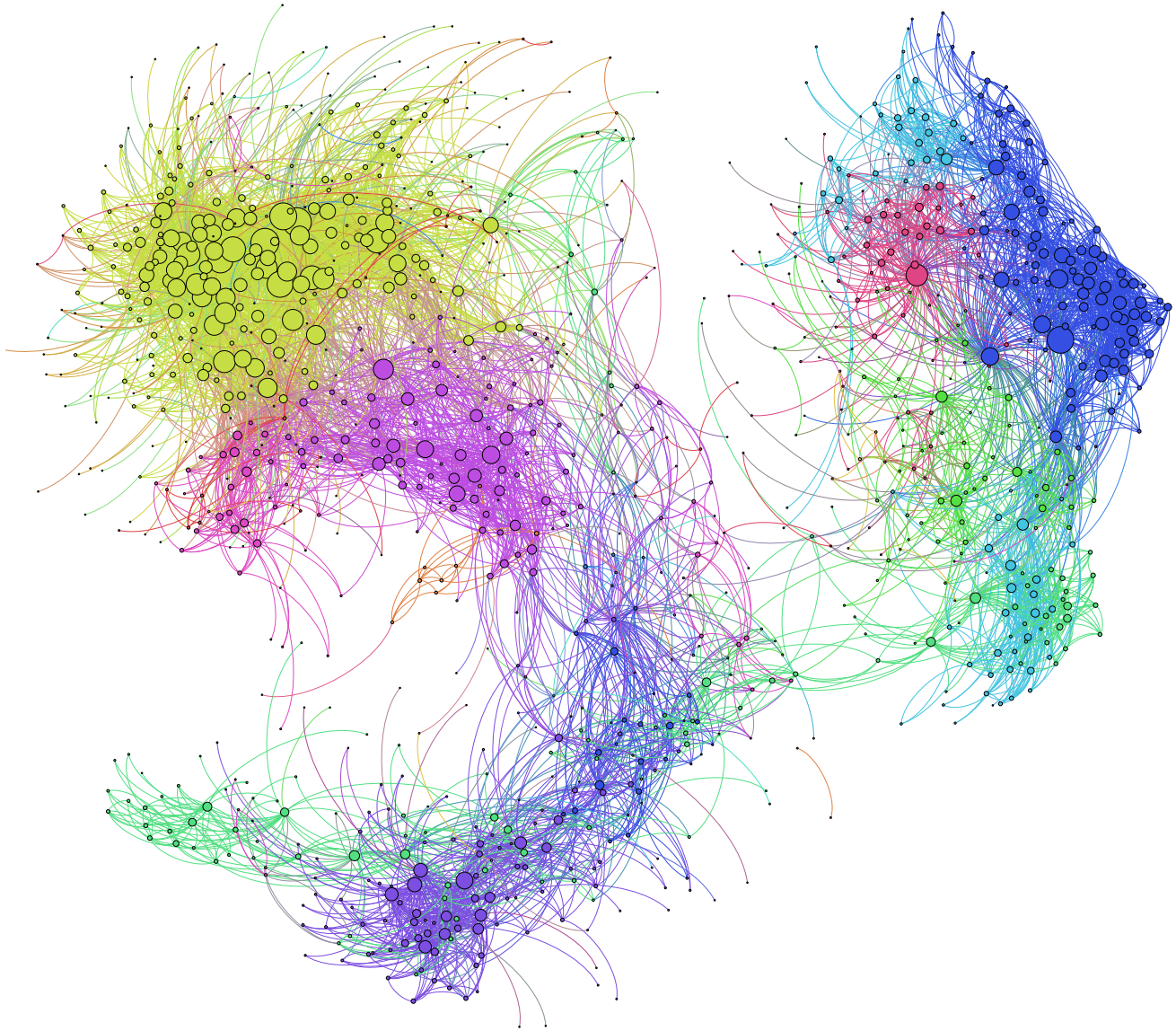
SIFT features can be use to

- align images
- rotate, scale, reshape images
- characterize and compare images: bag-of-SIFT-features
- Each image has 128 features



Network analysis

We want to compare nodes in a network. What features would be useful?



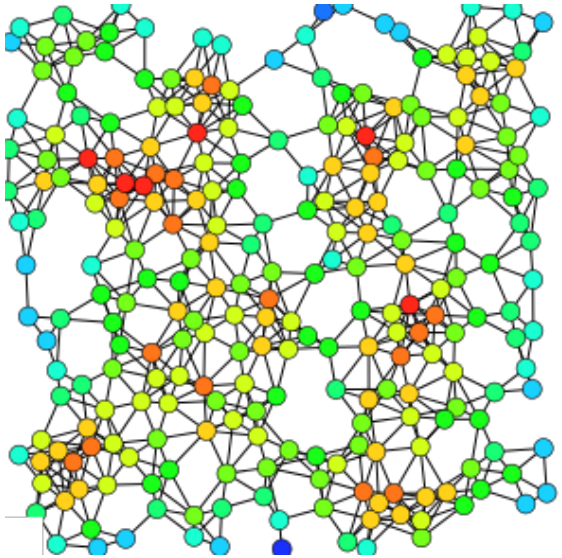
Network features

Node-specific features:

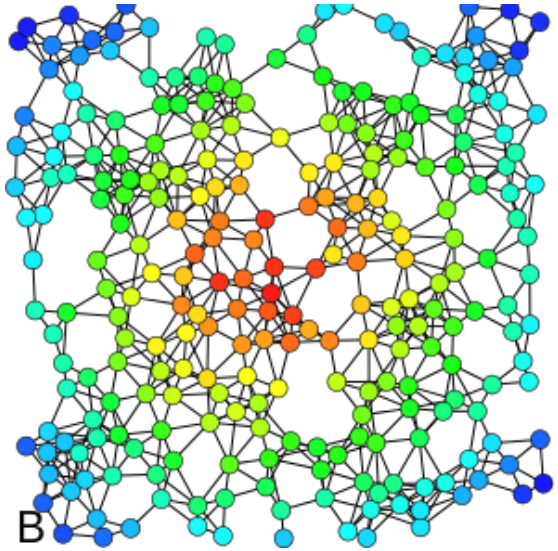
- Centrality:
  - *degree centrality*: average diff b/t largest degree node and node degree
  - *closeness centrality*: 1/(average shortest distance to all other nodes)
  - *betweenness centrality*: for all shortest paths between any pair of nodes, the number that pass through a specific node
- Hubs and Authorities [Kleinberg] ;
  - *hubs* catalog information
  - *authorities* are linked to by many pages
- many others...

Centrality: examples

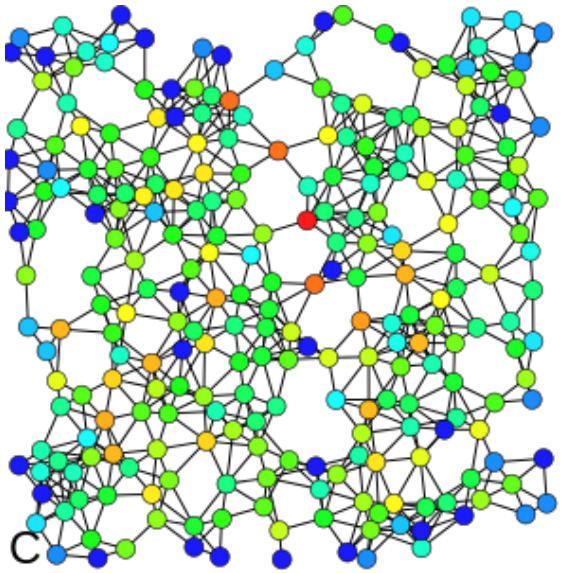
Degree centrality



Closeness centrality

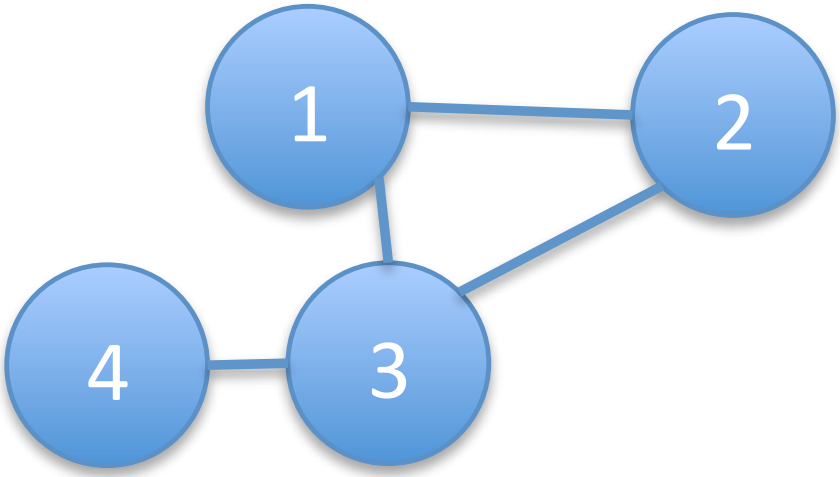


Betweenness centrality



Source: `wikipedia.org`

Node adjacency matrix



	1	2	3	4
1	0	1	1	0
2	1	0	1	0
3	1	1	0	1
4	0	0	1	0

Examples: Facebook, protein-protein interactions, Twitter, BitCoin transactions.

Encoding a network as an adjacency matrix

Adjacency matrix A: may be directed (non-symmetric) or undirected (symmetric); weighted or unweighted

- Adjacency can be thought of as a bag-of-neighbors: for every node, the number of features is the number of nodes, and counts are edges
- Eigenvalues of this matrix (e.g., PageRank [Page, Brin]) represent linear combinations of connected nodes, scaled by inverse path length

How can we compare networks, instead of nodes in a network?

Features: summary

- *Feature extraction* or *feature engineering* is an art, not a science
- Carefully-constructed features can mean the difference between a useful analysis and a failed analysis
- Some features will be predictive; other features will not
- Features may be correlated
- The best features will be informed by
  - the data domain
  - the analysis task
- Prune extraneous features: *Feature selection*. Why?

Kernel Functions

Kernels are a way to flexibly compare samples in a complex space.

Kernels have shown great utility in comparing:

- images of different sizes
- protein sequences of different lengths
- object 3D structures
- networks with different numbers of edges or nodes
- text documents of different lengths and formats.

Why Kernels?

Often objects are difficult to compare on the basis of their features alone.

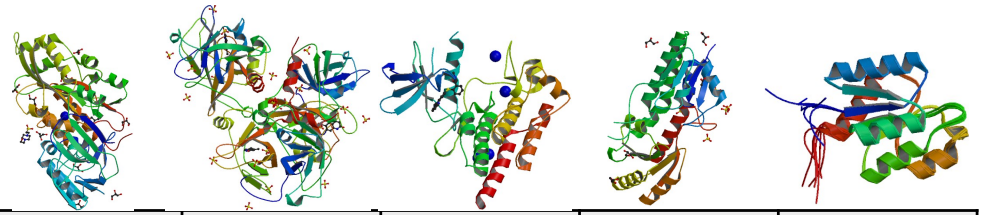
What features would you use to classify:

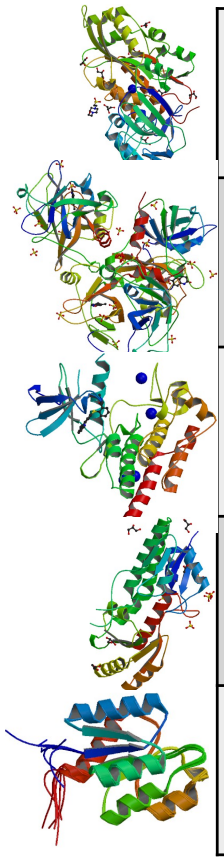
- 3D structures (e.g., molecules, proteins)
- time series data (e.g., stock prices)
- strings of unequal length (e.g., DNA sequences)
- network structures across different sets of random variables (e.g., evolutionary trees)
- many more...

Kernels are similarity functions that bypass feature representations.

Kernels for comparing samples

A well-defined kernel gives us a single metric to quantify the similarity between two samples





1	0.32	0.23	0.62	0.44
0.32	1	0.57	0.19	0.08
0.23	0.57	1	0.81	0.29
0.62	0.19	0.81	1	0.73
0.44	0.08	0.29	0.73	1

Kernels for classification

Classifying or clustering samples

- Simple classifiers may not perform well for a set of features.
- Kernels project features to a (higher dimensional) *feature space*
- Classifiers may work better in the kernelized feature space.

When describing the naive Bayes classifier, we used a fixed set of features.

Next lecture, we will describe classification methods that use kernels.

Kernel function

Given some abstract space $\mathcal{X}$ (e.g., documents, images, proteins, etc.), function $\kappa : \mathcal{X} \times \mathcal{X} \mapsto \Re$ is called a **kernel function**.

Kernel functions quantify similarity between two samples $\mathbf{x}$ and $\mathbf{x}'$ in $\mathcal{X}$.

For a given feature vector $\phi(\mathbf{x})$, we can construct a naive kernel:

$$\kappa(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^{\mathbf{T}} \phi(\mathbf{x}').$$

Now the set of features for one sample is one row of the kernel matrix:

$$[\kappa(x, x_1), \ldots, \kappa(x, x_n)].$$

Kernel function properties

A kernel function *may* or *may not* satisfy these two properties:

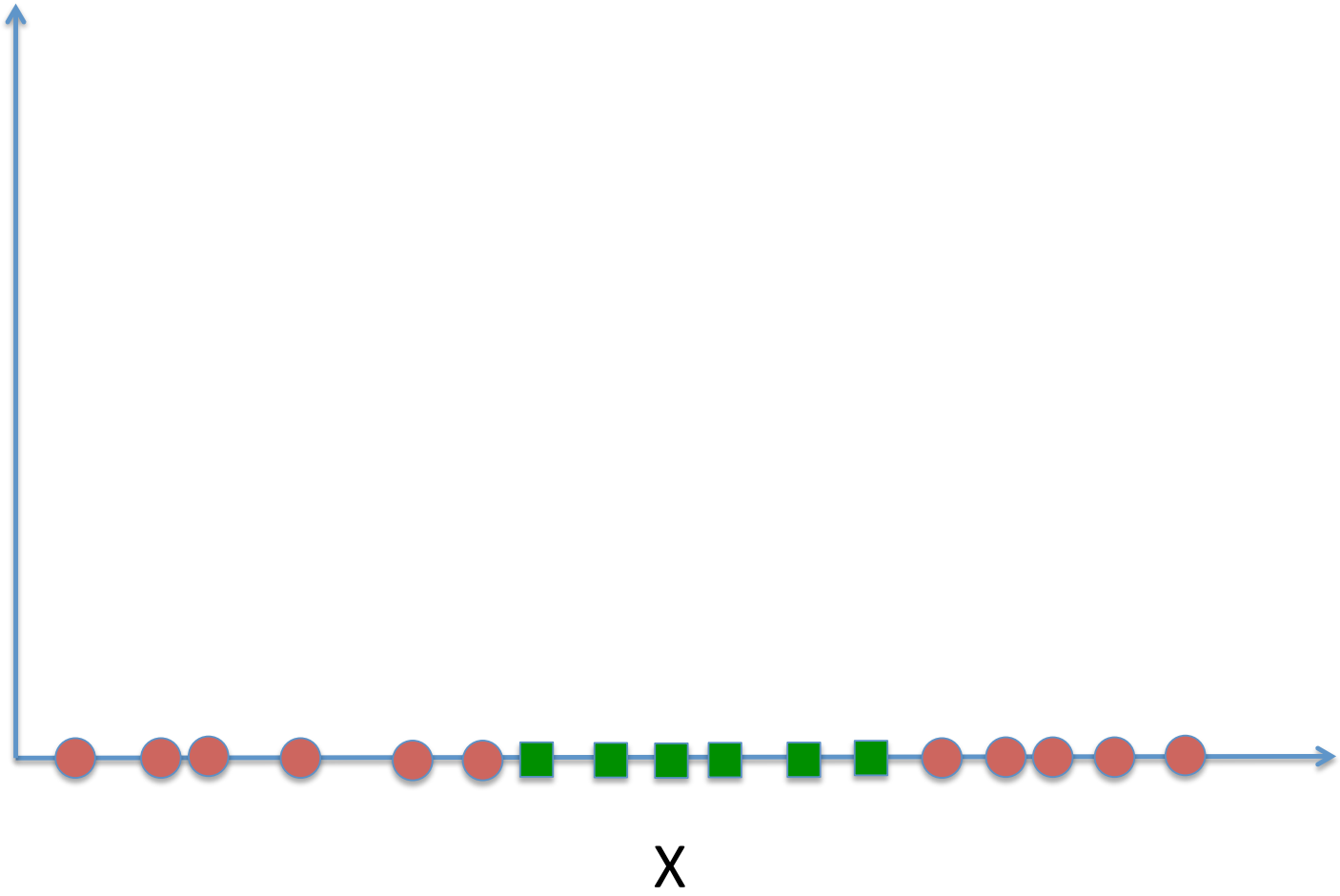
(symmetric) $\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}$, $\kappa(\mathbf{x}, \mathbf{x}') = \kappa(\mathbf{x}', \mathbf{x})$,

(non-negative) $\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}$, $\kappa(\mathbf{x}, \mathbf{x}') \geq \mathbf{0}$.

A kernel with these properties will loosely have the interpretation as a *similarity quantification* between the two samples.

Example: Kernels for classification

Let's say we have n scalar samples (i.e., a one dimensional input space) and a linear classifier.

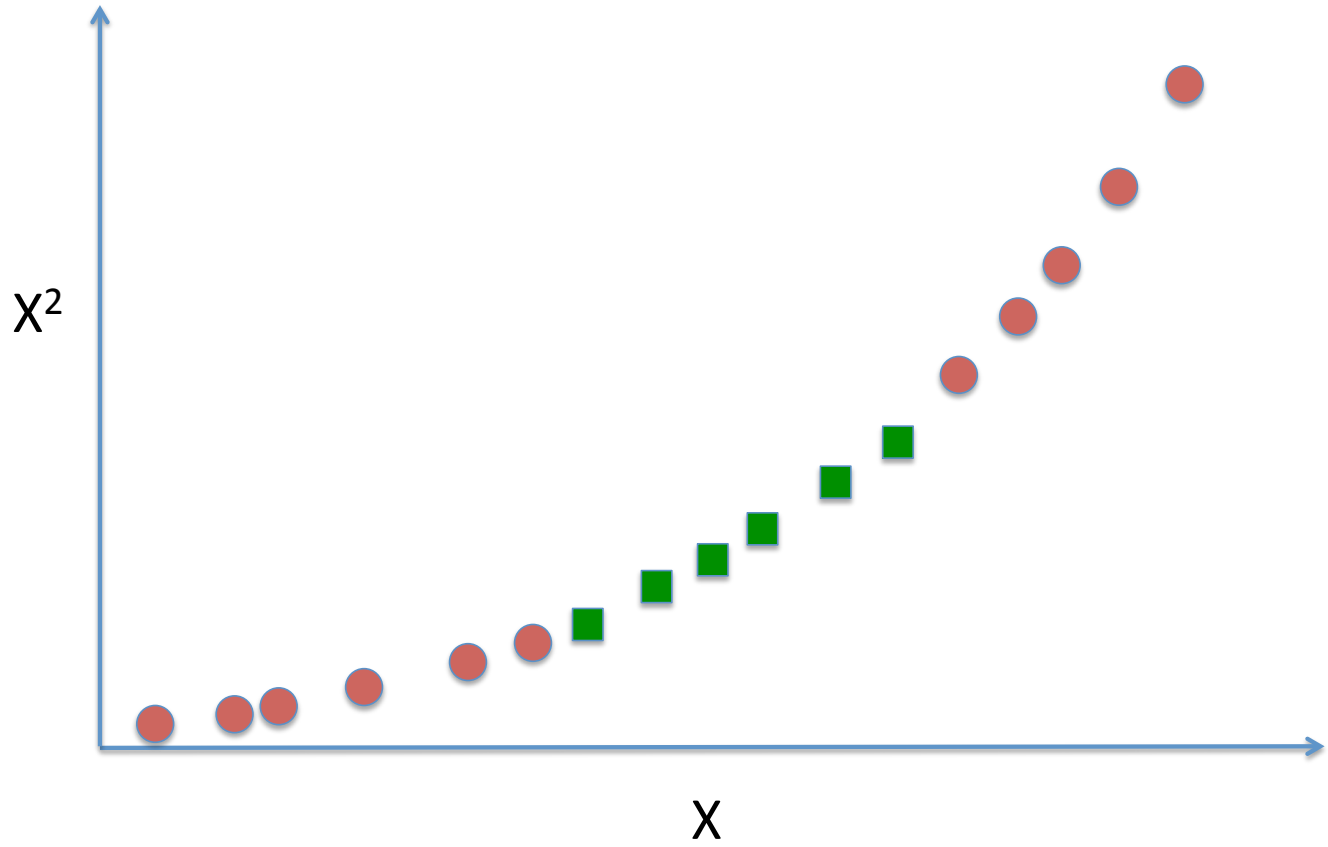


Can we separate the two classes?

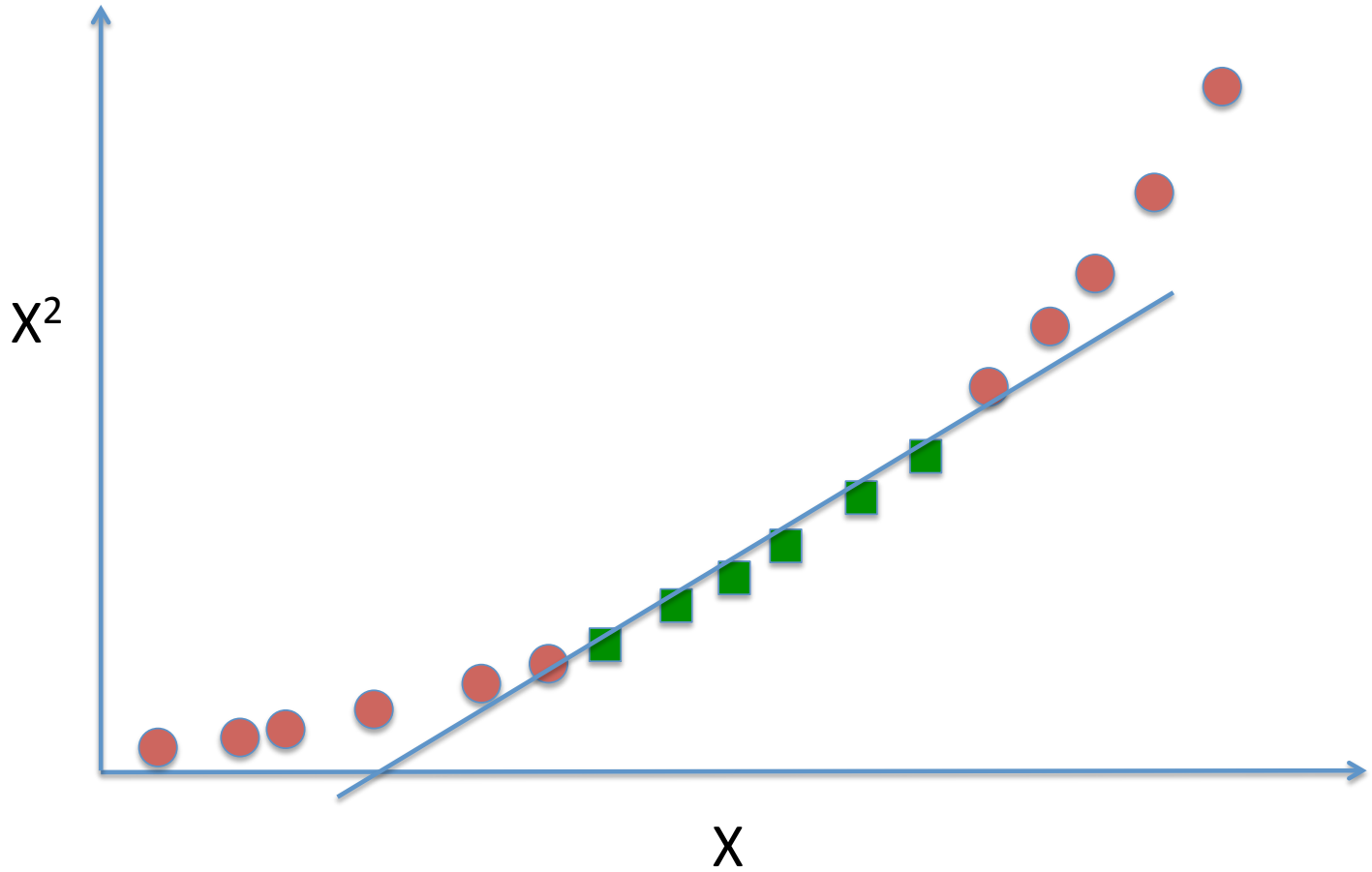
Example: Kernels for classification

Kernel solution: project features to a higher dimension, and use a linear classifier in projected *feature space*.

Let $\phi(\mathbf{x_i}) = [\mathbf{x_i}, \mathbf{x_i^2}]$; then $\kappa(\mathbf{x_i}, \mathbf{x_j}) = \phi(\mathbf{x_i})^\mathsf{T}\phi(\mathbf{x_j}) = \mathbf{x_i x_j} + \mathbf{x_i^2 x_j^2}$.



Example: Kernels for classification



If we project to x^2, will these samples be linearly separable?

Useful kernels

Let's talk about five useful types of kernels:

- Linear kernel
- Gaussian kernel
- Mercer kernels
- String kernel
- Fisher kernels

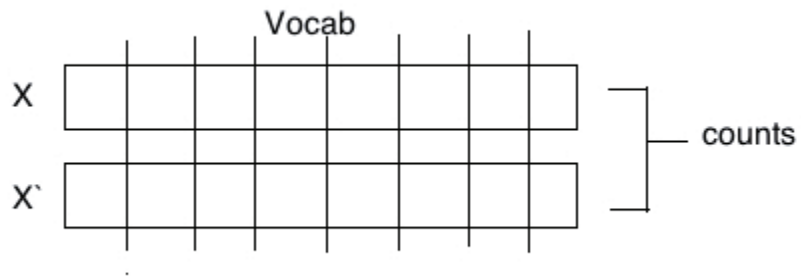
Linear kernels

Letting $\phi(\mathbf{x}) = \mathbf{x}$, we get the **linear kernel**, defined by the inner product between the two feature vectors:

$$\kappa(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\mathsf{T}\mathbf{x}'$$

The *dimension of the feature space D of a linear kernel* is the dimension of the input space $\mathcal{X}$, or the number of features of each sample

A linear kernel is useful when it is not necessary to perform an analysis in an alternative feature space (e.g., bag-of-words representations)



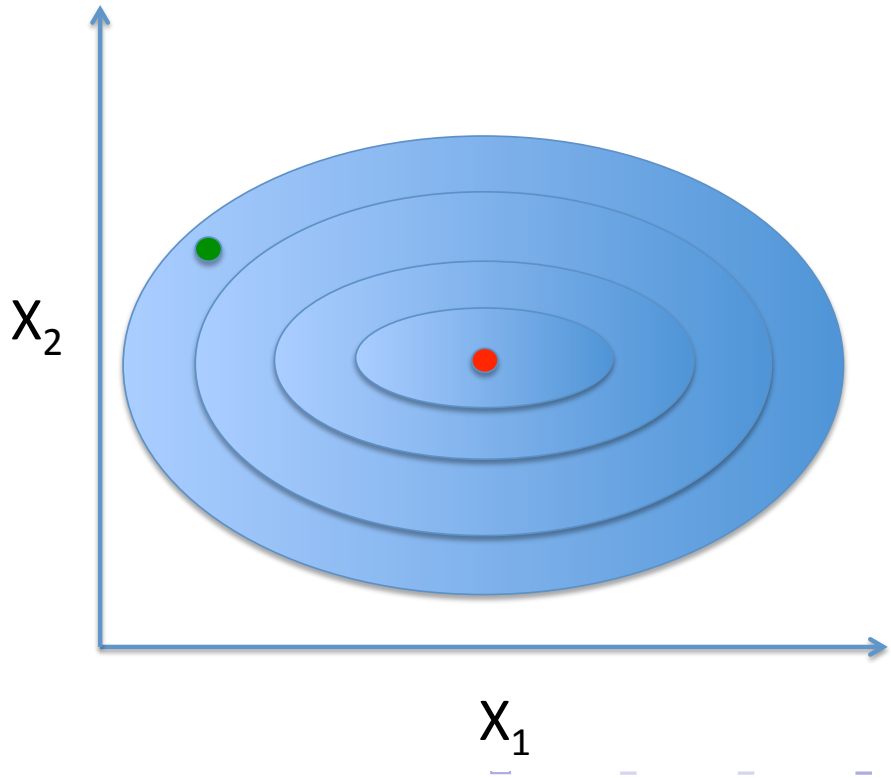
Gaussian Kernels

The *Gaussian kernel*, also known as the *squared exponential kernel* (SE) or *radial basis function* (RBF), is defined by

$$\kappa(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp\left(-\frac{1}{2}\sum_{j=1}^{p}\frac{1}{\ell_j^2}(x_j - x_j')^2\right)$$

This kernel has parameters σ^2 (output variance), and ℓ_j (characteristic length scale), for feature j.

Ex: height (x_1); shoe size (x_2)



Gaussian kernels: the kernel trick

The *dimension of the feature space D of a Gaussian kernel* is $|D| = \infty$: we can write this kernel as an infinite arithmetic series

$$\kappa(x, x') = \phi(x)^T \phi(x') = \sum_{j=1}^{\infty} \phi_j(x)\phi_j(x') = \exp\left(-\frac{1}{2}\sum_{j=1}^{p} \frac{1}{\sigma_j^2}(x_j - x_j')^2\right)$$

We never explicitly represent each sample in infinite *feature space*.

Instead, we compute the kernel in this infinite dimensional feature space with the kernel function; the projection is implicit.

Kernel trick

Computations are $O(n \times n)$, but feature space has arbitrarily high dimension.

Is every data set linearly separable in $|D| = \infty$?

Mercer Kernels

A *Mercer kernel* is defined by the inner product between two p-vectors:

$$\kappa(x, x') = \phi(x)^T \phi(x') = \sum_{j=1}^{p} \phi_j(x)\phi_j(x').$$

For $\{x_1, \ldots, x_n\}$, we can define a *Gram matrix*, which is the $n \times n$ matrix K with entries $K_{i,j} = \kappa(x_i, x_j)$.

The Gram matrix from a Mercer kernel is:

- positive semi-definite (all eigenvalues are nonnegative)
- symmetric (i.e., $\mathcal{K} = \mathcal{K}^T$).

Mercer Kernels: examples

Mercer kernel examples

- Linear kernel: $\phi(x) = x$
- Polynomial kernels, e.g., $\phi(x) = [x, x^2, x^3]$
- Gaussian kernel *[Schoelkopf & Smola 2002]*
- Sums of Mercer kernels, e.g., $\kappa(x, x') = \kappa_1(x, x') + \kappa_2(x, x')$

String Kernels

As an alternative to bag-of-words representation, we might be interested in finding all substrings of any pair of *strings*, or sequences of characters.

We can use a *string kernel*:

x: The quick brown fox jumped over the lazy dog.
x': Yesterday I went to town and saw a dove.

$\kappa_3(x, x') = 2$

String kernels: definition

- Let A denote an alphabet, e.g., $A = \{a, ..., z\}$
- Define $A^* = [A^1, A^2, \ldots, A^m]$, where m is the length of the longest substring to match

- Notation: superscripts are regular expression operators.
- A^i is the set of all possible strings of length i with any character from alphabet A
- * is known as the *Kleene star* operator.

String kernels: definition

- Basis function $\phi(\mathbf{x})$ maps a string $\mathbf{x}$ to a vector of length $|A^*|$
- Each element is the number of times we observe substring A_j^* in $\mathbf{x}$
- The string kernel is the weighted sum over all substrings in A^*:

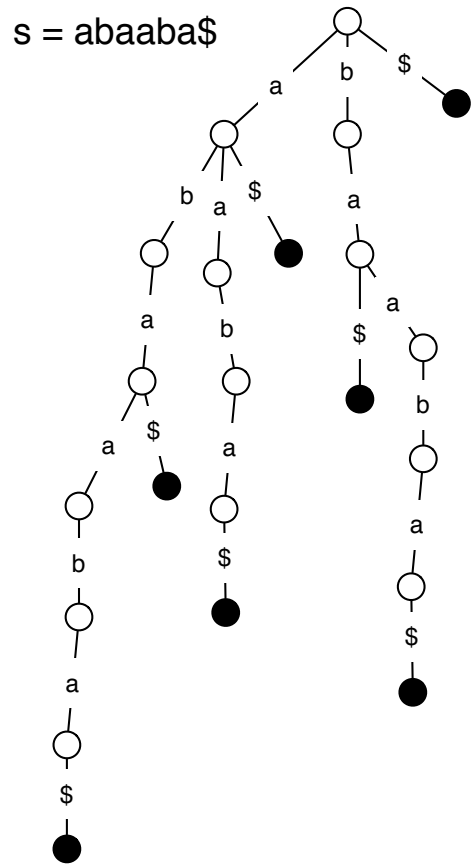
$$\kappa(x, x') = \sum_{s \in A^*} w_s \phi_s(x) \phi_s(x')$$

How can we avoid computations in the dimension of the feature space?

In practice, string kernels use a suffix tree

String kernel $\kappa(x, x')$ can be computed in $O(|x| + |x'|)$, or linear time, for weight function w, by only considering existing substrings.

- A *suffix tree* contains all possible suffixes in a string, condensed
- To compute $\kappa_m(\mathbf{x}, \mathbf{x}')$, build a m level suffix tree for string $\mathbf{x}$, and find all suffix matches with string $\mathbf{x}'$
- This is a linear time process.



Carl Kingsford suffix trie slides

Specialized string kernels

Customize a string kernel by setting weights w_s to specific values.

Specialized string kernels

- $w_s = 0$ for $|s| > 1$: comparing the alphabet between strings (i.e., substrings of length one)
- $w_s = |s|$: weight matches by the length of the matched substring
- $w_s = 0$ for s outside of a dictionary: equivalent to (weighted) bag-of-words kernel

Fisher Kernels

We can construct a kernel based on an *arbitrary generative model* using the concept of a *Fisher kernel*.

A Fisher kernel represents the distance in likelihood space between pairs of samples for a fitted generative model.

Fisher kernel

A *Fisher kernel* is the inner product of the gradient of the likelihood of x, x' given the fitted model scaled by inverse *Fisher information matrix*, or the information contained in observation x about model parameters θ.

Fisher Kernels allow arbitrarily complex generative models to capture the similarity between pairs of samples (e.g., Gaussian mixture model).

Summary: features and kernels

- Crafting informative features is essential to data analysis
- This is more art than science
- It is often difficult to identify predictive features of a complex sample
- Kernel functions and the kernel trick allow similarity between samples to be used as features in a computationally efficient way
- Other approaches: Word2vec, convolutional autoencoders
- These automated approaches lack interpretability

Additional Resources

- *MLAPA*: Chapter 14
- *Gaussian Processes for Machine Learning*: Chapter 4
- *Elements of Statistical Learning*: Chapter 3

- (video) Partha Niyogi: *Introduction to Kernel Methods*
- (video) Alex Smola: *Kernel Methods and Support Vector Machines*
- Metacademy: *The kernel trick*
- **The Kernel Cookbook**:
  `http://people.seas.harvard.edu/~dduvenaud/cookbook/`