Welcome to

Unsupervised Machine Learning

Kevyn Collins-Thompson

Associate Professor of Information and Computer Science
School of Information, University of Michigan

What is Unsupervised Learning?

- Supervised learning algorithms are ‘taught’ from labeled training examples.

- Unsupervised learning algorithms aren’t given any labeled data!

Why this course?

1. Cover critical basic principles used in all future machine learning-related courses:
    - Dimensionality reduction
    - Clustering
    - Topic models and text processing
2. Broad coverage of unsupervised algorithms
    - PCA and non-negative matrix factorization
    - Clustering: k-means, hierarchical, and more
    - Topic modeling, latent semantic indexing

Key aspects of Unsupervised Learning

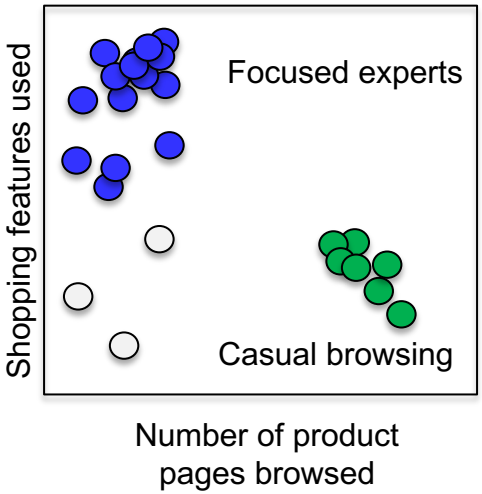
For a set of unlabeled data instances:

- Transform or summarize
- Find clusters or other interesting structure

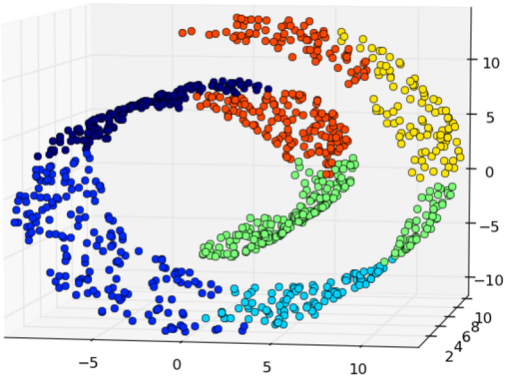
There is no target value to be predicted.

Input: Feature representation (from indiv. features)

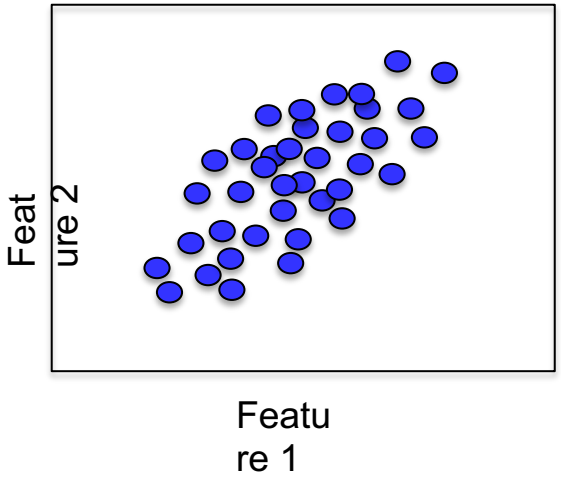
What is interesting structure?



Global cluster structure

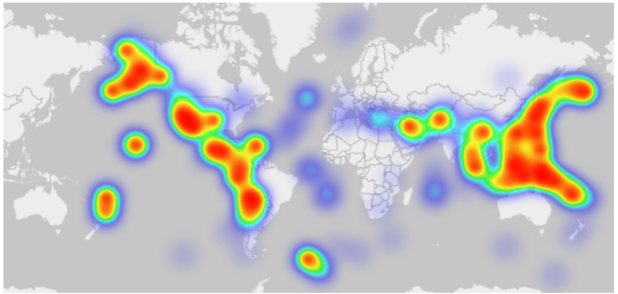


Local neighborhood structure

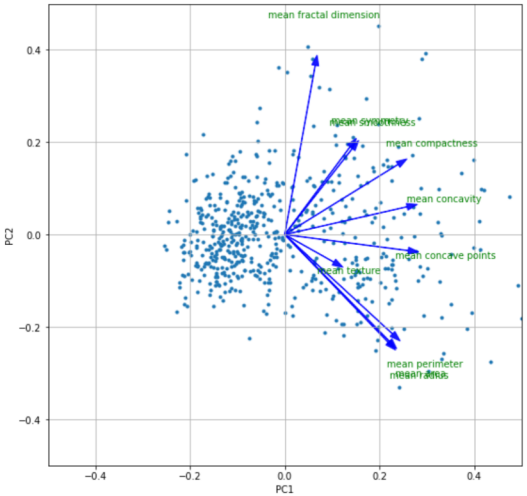


Correlation structure

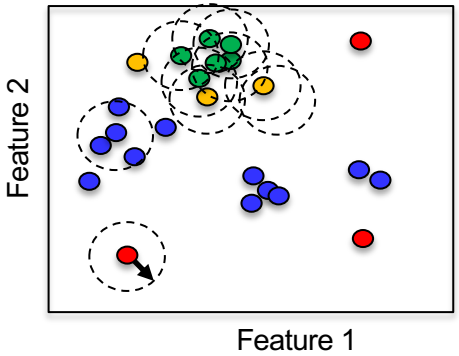
Some examples of unsupervised learning



Density estimation
(probability of earthquake activity)



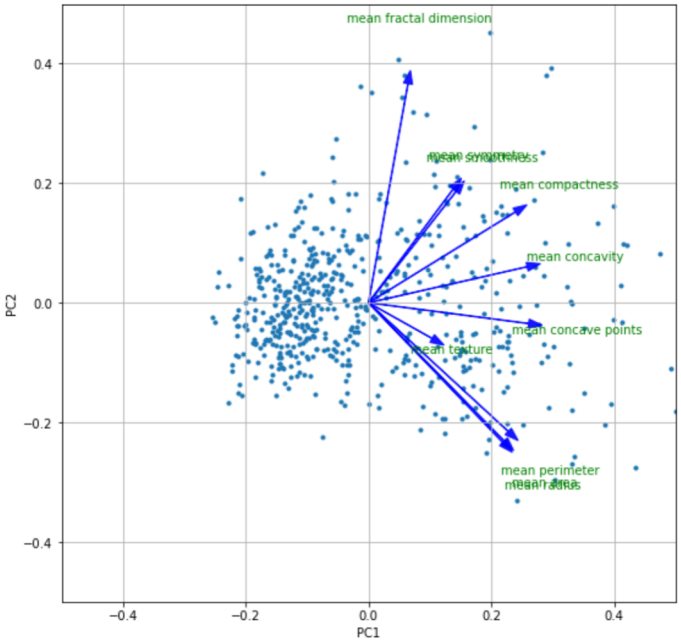
Dimensionality reduction



Clustering

Source: http://www.digital-geography.com/csv-heatmap-leaflet/

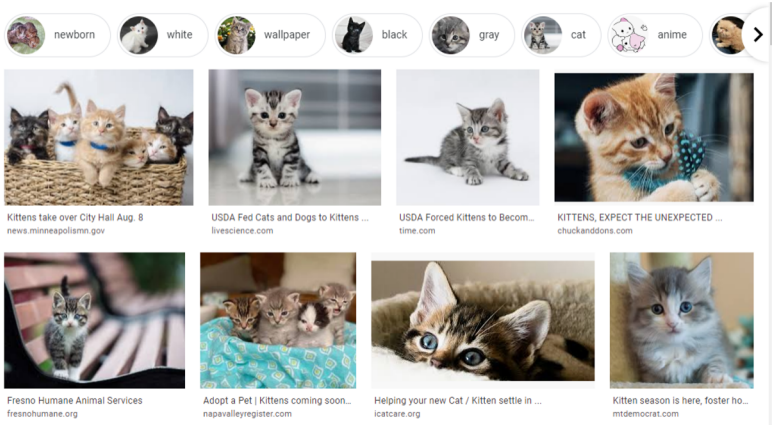
Dimensionality reduction

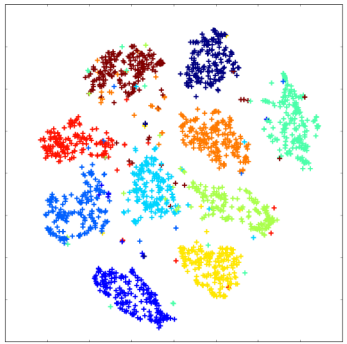


Correlated properties of cancer cells

- Makes it easier to work with high-dimensional data.
- Get insight into correlated variables.
- Useful for compression.
- Useful for obtaining new features for supervised learning.

Clustering





- Clustering partitions objects into groups that share similar features
- Items within a group are similar
- Items in different groups are dissimilar
- Examples:
  - Image search
  - Web traffic analysis
  - … and many more

Jiang, Zheng, Tan, Tang, Bangsheng, Zhou. (2017). Variational Deep Embedding: An Unsupervised and Generative Approach to Clustering. 1965-1972. 10.24963/ijcai.2017/273.

Topic modeling

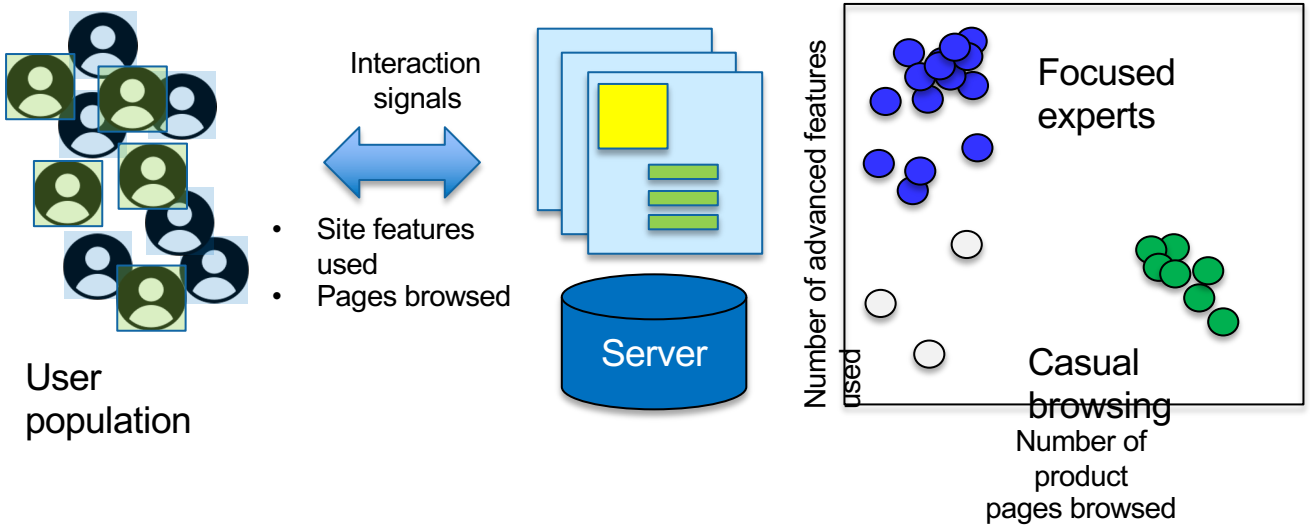
Topic 1	Topic 2	Topic 3	Topic 4	Topic 5
computer	chemistry	cortex	orbit	infection
methods	synthesis	stimulus	dust	immune
number	oxidation	fig	jupiter	aids
two	reaction	vision	line	infected
principle	product	neuron	system	viral
design	organic	recordings	solar	cells
access	conditions	visual	gas	vaccine
processing	cluster	stimuli	atmospheric	antibodies
advantage	molecule	recorded	mars	hiv
important	studies	motor	field	parasite

FIGURE 1. Five topics from a 50-topic LDA model fit to *Science* from 1980–2002.

Terms are usually ordered by their probability given the topic.

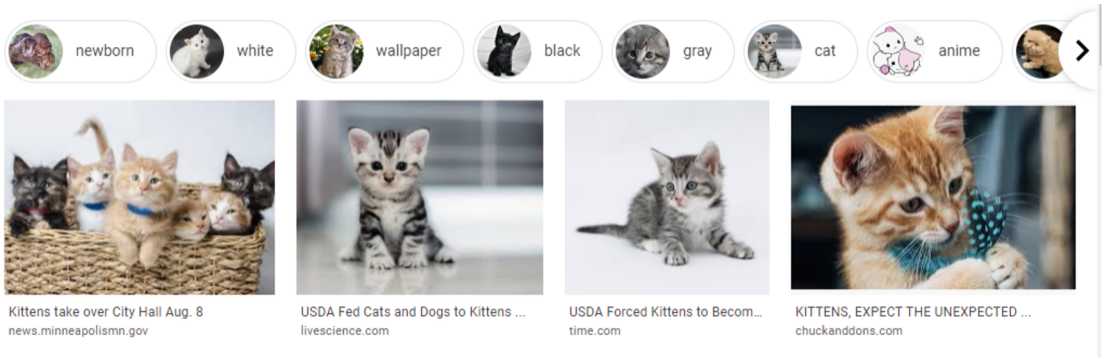
Source: Blei & Lafferty (2009) “Topic Modeling”, http://www.cs.columbia.edu/~blei/papers/BleiLafferty2009.pdf

Example: Clustering web traffic



Challenges of unsupervised learning

- Evaluating output quality can be difficult
  - We have no ‘ground truth’ labels!
  - Infinite number of ways to group kittens: which ones are ‘best’?



Challenges of unsupervised learning

- Interpreting clusters or discovered structure can be difficult
  - Often needs human intelligence

computer	chemistry	cortex
methods	synthesis	stimulus
number	oxidation	fig
two	reaction	vision
principle	product	neuron
design	organic	recordings
access	conditions	visual
processing	cluster	stimuli
advantage	molecule	recorded
important	studies	motor

What should these topics be called?

Learning objectives for this course

After you take this course in unsupervised learning, you will:

- Be able to correctly apply and interpret results from clustering methods in scikit-learn, including k-means, agglomerative clustering, hierarchical clustering, and DBSCAN.
- Understand the use of topic modeling and best practices for its application.
- Be able to correctly apply and interpret results from manifold learning methods, including multidimensional scaling and t-SNE.
- Understand how to evaluate clustering results using a variety of metrics.

Learning objectives

- Understand the tradeoffs and assumptions inherent in different clustering techniques.
- Understand how unsupervised learning can be used to improve supervised prediction.
- Know how to perform density estimation using a kernel, with a single random variable.
- Know how to interpret a biplot result from principal components analysis (PCA).

Learning objectives: awareness

- Word embeddings
- The EM algorithm
- Advanced methods like spectral clustering.

This course will help prepare you for…

- Deep Learning
- Machine Learning Pipelines
- Milestone II project
- Natural Language Processing and many other domain-specific application courses

Course schedule

- Week 1: Dimensionality reduction, manifold learning, density estimation
- Week 2: Clustering
- Week 3: Topic modeling
- Week 4: Applications of unsupervised learning

Kevyn Collins-Thompson

kevynct@umich.edu

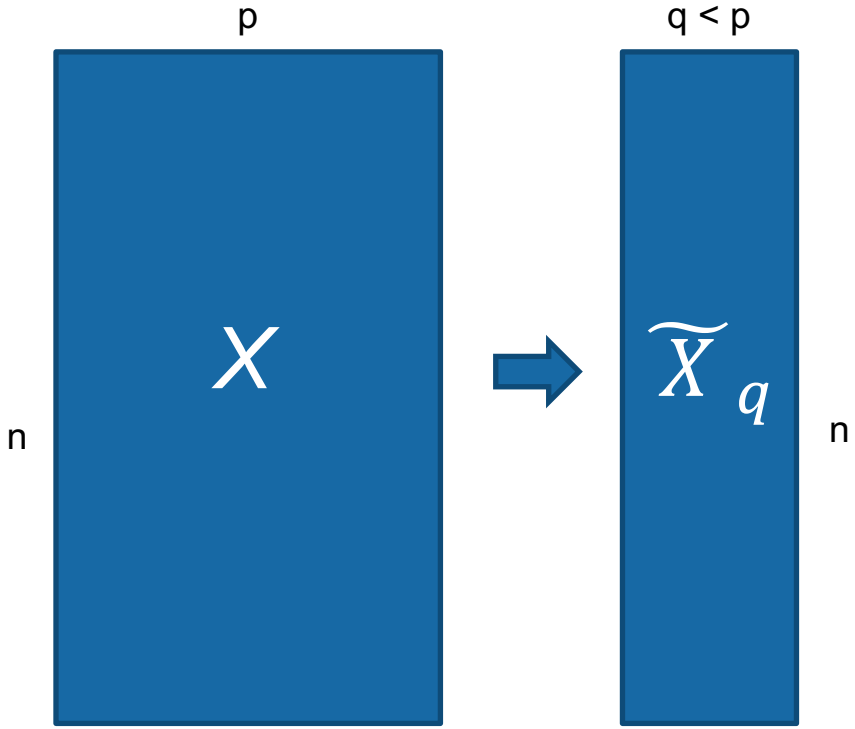
School of Information

Dimensionality Reduction: Principal Components Analysis (PCA)

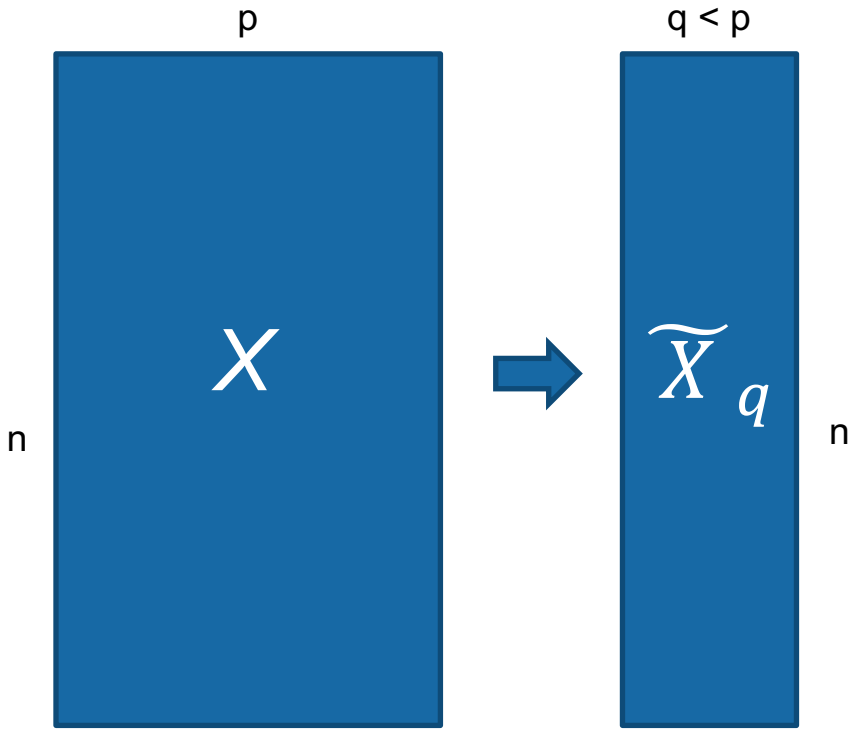
Kevyn Collins-Thompson

Associate Professor of Information and Computer Science
School of Information, University of Michigan

Dimensionality Reduction

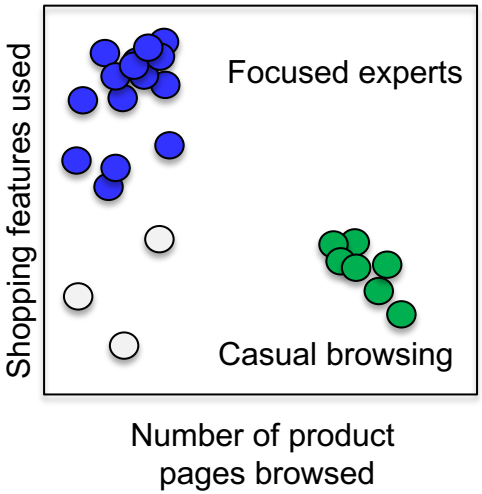


Dimensionality Reduction

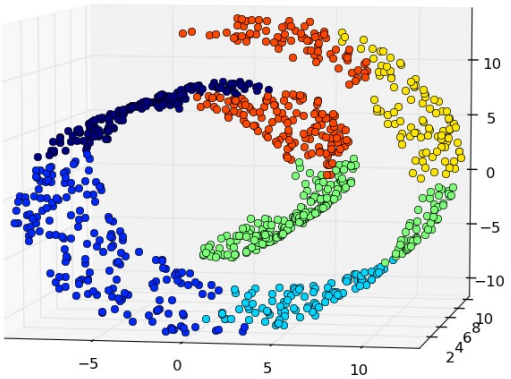


- Explore a dataset.
- Get around the ‘curse of dimensionality’ by finding an approximation with much lower dimension.
- More efficient/effective statistical tests or learning algorithms.
- Compress to save space.
- Find interesting structure to improve accuracy of supervised learning predictions.

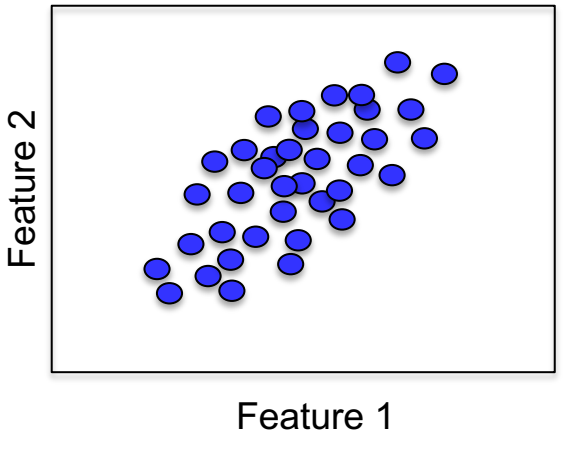
What is interesting structure?



Global cluster structure



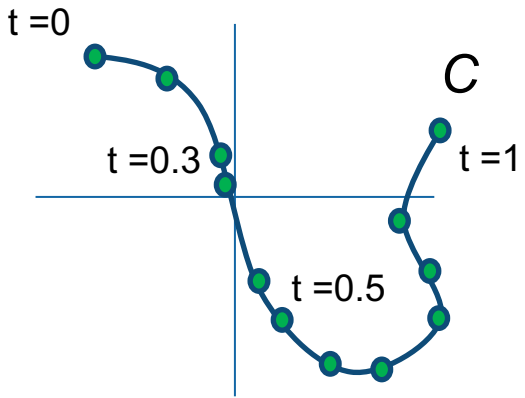
Local neighborhood structure



Correlation structure

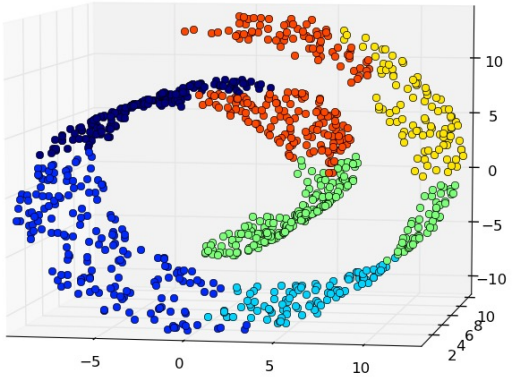
Intrinsic vs extrinsic dimensionality

Extrinsic dimensionality of data points: 2



Intrinsic dimensionality: 1

Given *C*, any point on *C* can be identified by a single unique number (time *t*)

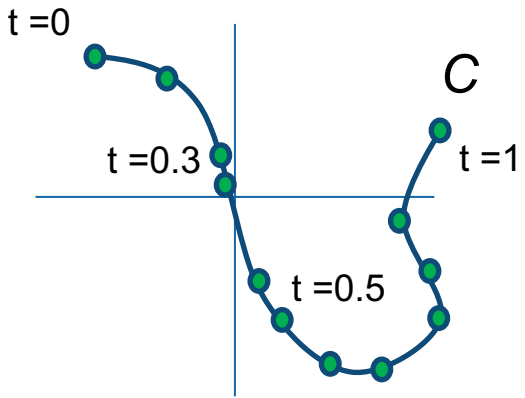


What is the extrinsic dimensionality?

What is the intrinsic dimensionality?

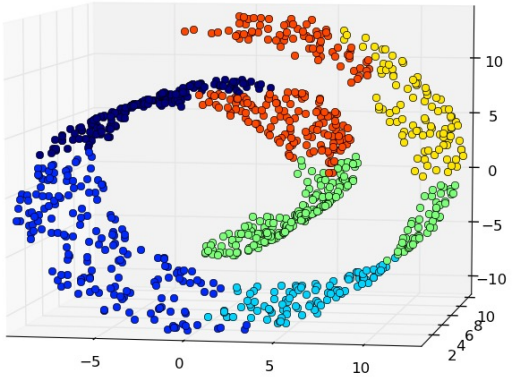
Intrinsic vs extrinsic dimensionality

Extrinsic dimensionality of data points: 2



Intrinsic dimensionality: 1

Given *C*, any point on *C* can be identified by a single unique number (time *t*)



What is the extrinsic dimensionality?

➡ Three

What is the intrinsic dimensionality?

➡ Two

Three main ways to reduce dimensionality of your dataset

- Feature elimination
- Feature selection
- Feature extraction

Three main ways to reduce dimensionality of your dataset

- Feature elimination
  - Select a subset of features (columns) arbitrarily
  - Useful for exploratory analysis, loses information
- Feature selection
- Feature extraction

Three main ways to reduce dimensionality of your dataset

- Feature elimination
  - Select a subset of features (columns) arbitrarily
  - Useful for exploratory analysis, loses information
- Feature selection
  - Score each feature (column) w/ statistical test
  - Rank the features by score, keep the top k
- Feature extraction

Feature selection: Information Gain

T is the label distribution

a is the feature in question

$S_a(v)$ = Labels for the subset of instances where attribute *a* takes value *v*

H(*T*) is the entropy of distribution T.

H(T|a) is the conditional entropy.

$$\mathrm{H}(T|a) = \sum_{v \in vals(a)} \frac{|S_a(v)|}{|T|} \cdot \mathrm{H}\left(S_a(v)\right).$$

$$IG(T,a) = \mathrm{H}(T) - \mathrm{H}(T|a)$$

Higher IG scores are better!

a	b	T
1	1	0
2	1	1
0	1	0
0	2	0
1	2	0
1	2	0
2	2	1

$$\mathrm{H}(\mathrm{T}|\mathrm{a}) = \frac{2}{7}H\big(S(0)\big) + \frac{3}{7}H\big(S(1)\big) + \frac{2}{7}H\big(S(2)\big)$$

$$\mathrm{H}\big(S(0)\big) = 0 \quad \mathrm{H}\big(S(1)\big) = 0 \quad \mathrm{H}\big(S(2)\big) = 0$$

$$\Rightarrow \quad \mathrm{H}(\mathrm{T}|\mathrm{a}) = 0$$

$$\mathrm{H}(T) = -\frac{5}{7}\log\frac{5}{7} - \frac{2}{7}\log 2/7$$

$= 0.346 + 0.516 = 0.862$ (log base 2 is used)

Therefore the IG of feature 'a' is:

$$\mathrm{H}(\mathrm{T}) - \mathrm{H}(\mathrm{T}|a) = 0.862 - \ 0 = \mathbf{0.862}$$

By similar calculations, IG of feature 'b' is:

$$\mathrm{H}(\mathrm{T}) - \mathrm{H}(\mathrm{T}|b) = 0.862 - 1.2 = \mathbf{-0.338}$$

So at least by IG score, feature 'a' is superior to feature 'b'
(in fact, you can verify it splits the label classes perfectly.)

Other feature selection metrics

Name	Description	Formula
Acc	Accuracy	$tp - fp$
Acc2	Accuracy balanced†	$\lvert tpr - fpr \rvert$
BNS	Bi-Normal Separation†	$\lvert F^{-1}(tpr) - F^{-1}(fpr) \rvert$ where F is the Normal c.d.f.
Chi	Chi-Squared‡	$t(tp, (tp+fp)P_{pos}) + t(fn, (fn+tn)P_{pos}) + t(fp, (tp+fp)P_{neg}) + t(tn, (fn+tn)P_{neg})$ where $t(count, expect) = (count - expect)^2 / expect$
DFreq	Document Frequency†‡°	$tp + fp$
F1	F_1-Measure	$\frac{2\,recall\ precision}{(recall + precision)} = \frac{2\,tp}{(pos + tp + fp)}$
IG	Information Gain†‡	$e(pos, neg) - [\, P_{word}\, e(tp, fp) + P_{\overline{word}}\, e(fn, tn)\,]$ where $e(x, y) = -\frac{x}{x+y}\log_2\frac{x}{x+y} - \frac{y}{x+y}\log_2\frac{y}{x+y}$
OddN	Odds Ratio Numerator	$tpr\,(1 - fpr)$
Odds	Odds Ratio†	$\frac{tpr\,(1-fpr)}{(1-tpr)\,fpr} = \frac{tp\ tn}{fp\ fn}$
Pow	Power	$(1 - fpr)^k - (1 - tpr)^k$ where k=5
PR	Probability Ratio	tpr / fpr
Rand	Random†°	random()

† Acc2, BNS, DFreq, IG, and Odds select a substantial number of negative features.
‡ Chi, IG, DFreq, and Rand also generalize for multi-class problems.
° DFreq and Rand do not require the class labels.

Notation:

tp: true positives = number of positive cases containing word
fp: false positives = number of negative cases containing word
pos: number of positive cases = *tp* + *fn*
neg: number of negative cases = *fp* + *tn*
tpr: sample true positive rate = *tp* / *pos*
fpr: sample false positive rate = *fp* / *neg*
precision = *tp* / (*tp*+*fp*)
fn: false negatives
tn: true negatives
P_{pos} = *pos* / *all*
P_{neg} = *neg* / *all*
P_{word} = (*tp*+*fp*) / all
$P_{\overline{word}}$ = 1-P(word)
recall = *tpr*

Note: Metrics such as BNS, Chi and IG are naturally symmetric with respect to negatively correlated features. For the metrics that devalue all negative features, we invert any negative feature, i.e. $tpr' = 1 - tpr$ and $fpr' = 1 - fpr$, without reversing the classes. Hence, without loss of generality, $tpr > fpr$.

Practical example:

George Forman, 'An Extensive Empirical Study of Feature Selection Metrics for Text Classification'

Journal of Machine Learning Research, 2003.

Source: https://www.jmlr.org/papers/volume3/forman03a/forman03a.pdf

Feature selection: subsets

- Wrapper methods search over the space of all possible subset of features
  - Sequential forward selection
  - Genetic algorithms
- Repeatedly call the learning algorithm on a validation set to evaluate feature subset
- Computationally intensive, difficult or impossible in higher dimensions

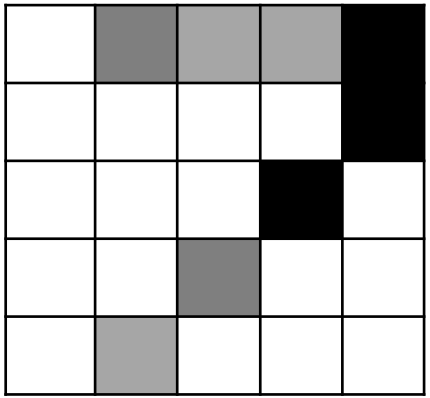
Three main ways to reduce dimensionality of your dataset

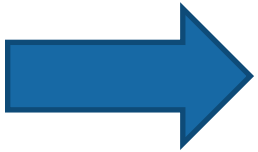
- Feature elimination
- Feature selection
- Feature extraction

Feature extraction

- Transform your dataset into a version with fewer columns, based on new features.
- These new features should:
  - Still capture the original variation and structure in the data
  - But are linearly independent
  - Be smaller in number than the old feature set
- Two main approaches:
  - Linear dimensionality reduction
    - New features are linear transforms of the old features
  - Non-linear dimensionality reduction
    - New features are non-linear transforms of the old features

Feature extraction: grey-scale digit images





7

25-dimensional feature space:
5x5 integer array, flattened

1-dimensional feature space:
1 integer

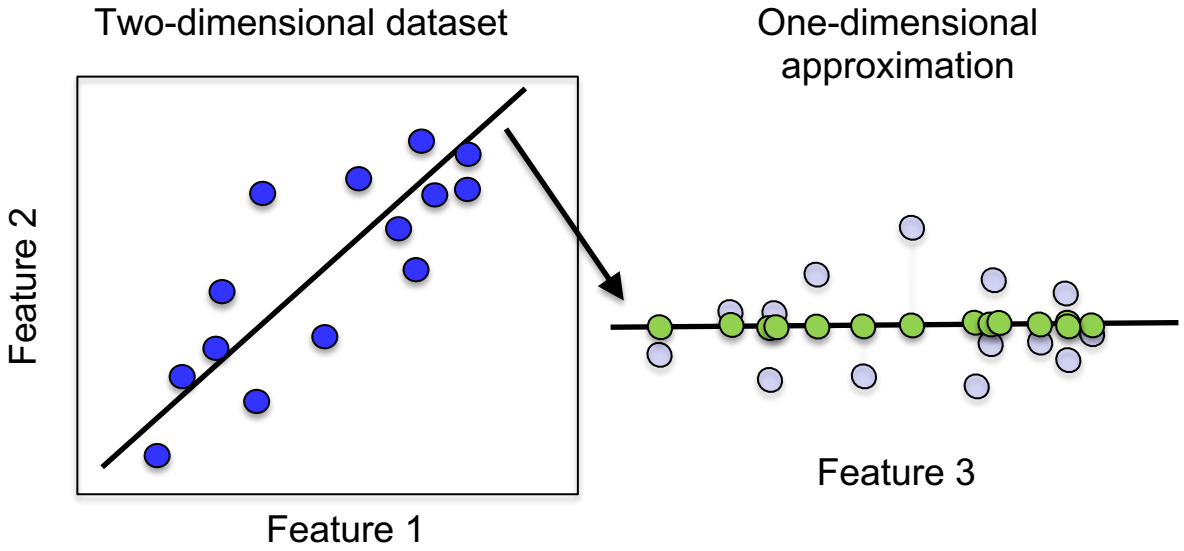
0 1 2 2 3 0 0 0 0 3 0 0 0 3 0 0 0 2 0 0 0 1 0 0 0

7

Bonus question: Is this linear or non-linear feature extraction?

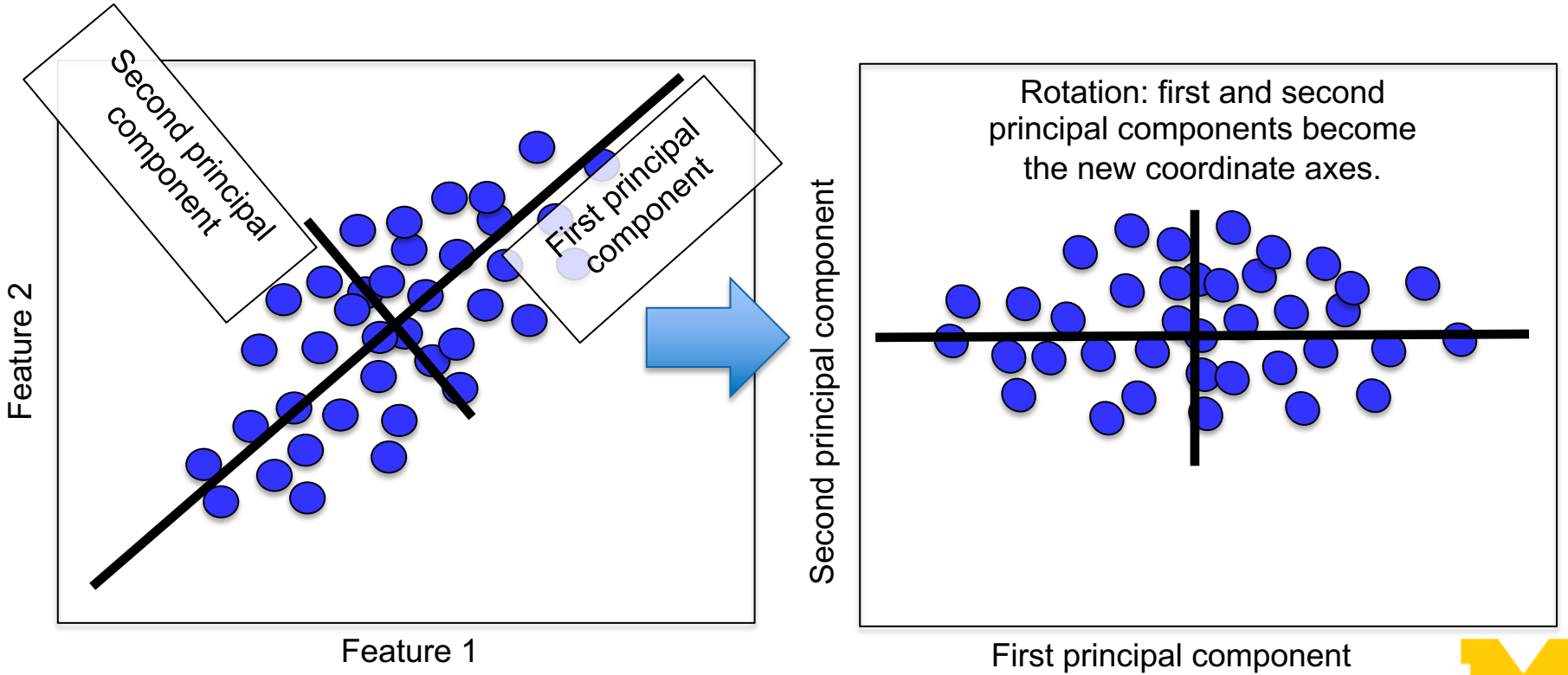
Dimensionality Reduction: Principal Components Analysis

- Finds an approximate version of your dataset using fewer features
- Used for exploring and visualizing a dataset to understand grouping or relationships
- Often visualized using a 2-dimensional scatterplot
- Also used for compression, finding features for supervised learning

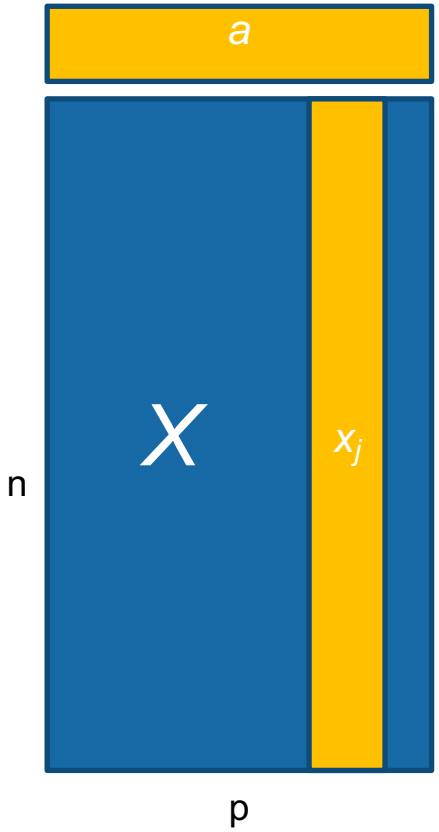


The one-dimensional approximation is obtained by projecting the original points onto the diagonal line and using their position on that line as the new single feature.

Simple PCA Example



Linear algebra basis for PCA



$$Xa = \sum_{j=1}^{p} a_j x_j$$

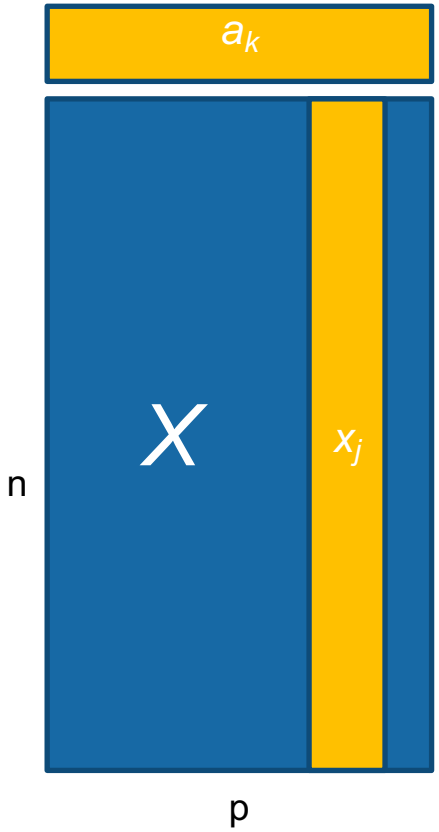
We want to find ***vector a*** that ***maximizes*** $var(Xa) = a^T Sa$ *subject to* $a^T a = 1$

$$\max_a \quad a^T Sa - \lambda(a^T a - 1) \qquad \text{(Equation 1)}$$

Differentiating we get: $Sa - \lambda a = 0$ or $Sa = \lambda a$ (Equation 2)

For eigenvector a with eigenvalue λ : $a^T Sa = \lambda a^T a = \lambda$

Linear algebra basis for PCA



PCA usually uses a centered version X* of matrix X as input.

$$x^*_{ij} = x_{ij} - x\_mean_j$$

After centering, the covariance matrix *S* can be expressed in terms of X*

$$(n-1)S = {X^*}^T X^*$$

The Singular Value Decomposition of a matrix Y is obtained from the eigendecomposition of Y^TY.
Hence PCA is equivalent to the SVD of the centered data matrix X*

(More on this and other details of SVD in a separate lecture)

PCA on the Wisconsin Breast Cancer dataset

mean radius
mean texture
mean perimeter
mean area
mean smoothness
mean compactness
mean concavity
mean concave points
mean symmetry
mean fractal dimension
radius error
texture error
perimeter error
area error
smoothness error
compactness error
concavity error
concave points error
symmetry error
fractal dimension error
worst radius
worst texture
worst perimeter
worst area
worst smoothness
worst compactness
worst concavity
worst concave points
worst symmetry
worst fractal dimension

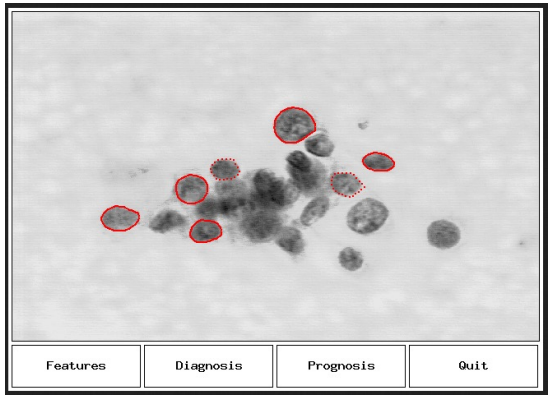


Image source: http://pages.cs.wisc.edu/~street/saves/xcyt1.gif

Applying PCA to the Wisconsin breast cancer dataset

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()
(X_cancer, y_cancer) = load_breast_cancer(return_X_y = True)

# Before applying PCA, each feature should be centered (zero mean) and with unit variance
X_cancer_normalized = StandardScaler().fit(X_cancer).transform(X_cancer)

pca = PCA(n_components = 2).fit(X_cancer_normalized)
X_pca = pca.transform(X_cancer_normalized)
print(X_cancer_normalized.shape, X_pca.shape)
```

```
(569, 30) (569, 2)
```

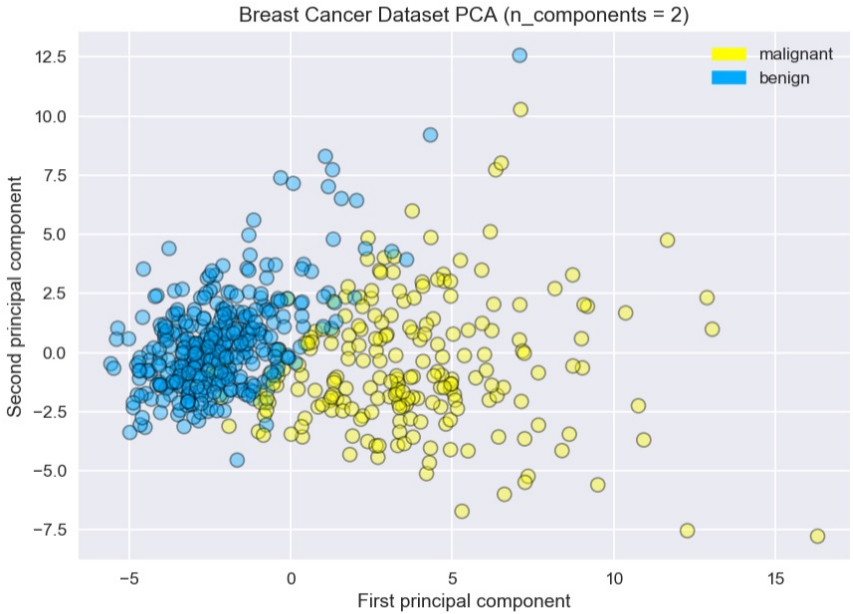
Important! Preparing your data for input to PCA

- By default, scikit-learn PCA will column-center the data, but not scale it!
- Normally you should always scale the data so that each variable has zero mean, unit variance.
- You can use the StandardScaler class before calling PCA, or the ‘whiten’ PCA option.
- Exception: when the units of all variables are the same, scaling is not advised, because it shrinks features containing strong signals and inflates features with little/no signal.
- As with all forms of normalization or scaling, if you’re using the results for supervised learning, ALWAYS do the train/test split FIRST, before normalization, PCA or anything else!

Applying PCA to the Wisconsin breast cancer dataset

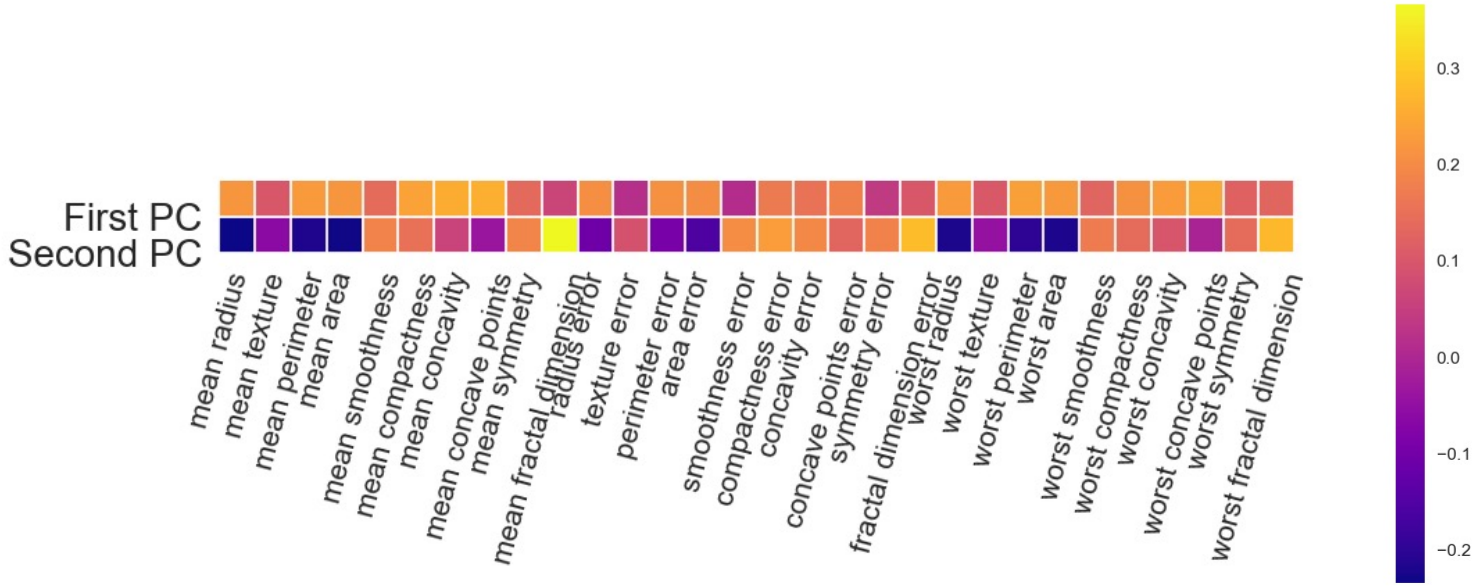
```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()
(X_cancer, y_cancer) = load_breast_cancer(return_X_y = True)

# Before applying PCA, each feature should be centered (zero mean) and with unit variance
X_cancer_normalized = StandardScaler().fit(X_cancer).transform(X_cancer)

pca = PCA(n_components = 2).fit(X_cancer_normalized)
X_pca = pca.transform(X_cancer_normalized)
print(X_cancer_normalized.shape, X_pca.shape)
```

```
(569, 30) (569, 2)
```



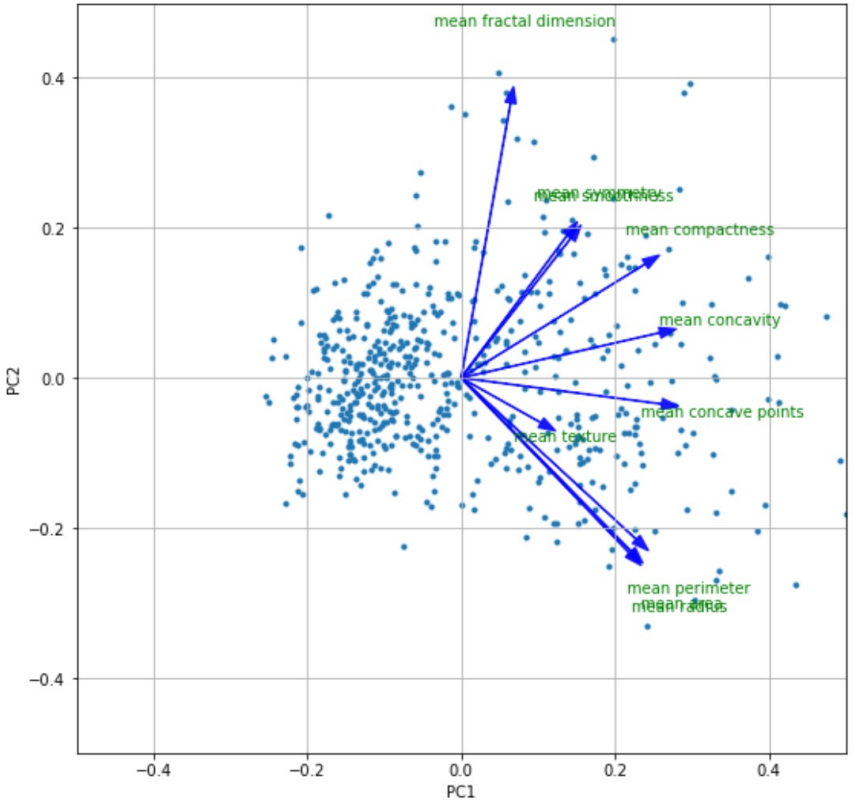

```
from adspy_shared_utilities import plot_labelled_scatter
plot_labelled_scatter(X_pca, y_cancer, ['malignant', 'benign'])

plt.xlabel('First principal component')
plt.ylabel('Second principal component')
plt.title("Breast Cancer Dataset PCA (n_components = 2)")
```

Visualizing PCA Components



Biplots

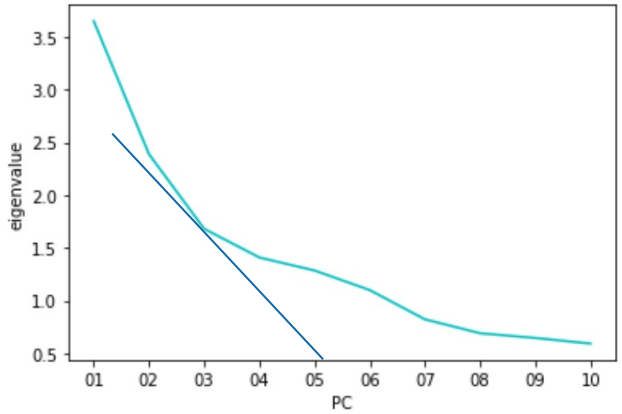
- Combines a score plot and a loadings plot in a single graph.
- Points show the projected observations.
- Vectors are the projected variables.
- If the data are well-approximated by the first two principal components, a biplot enables you to visualize high-dimensional data by using a two-dimensional graph.
- See the separate video.



Scree plot for choosing top components

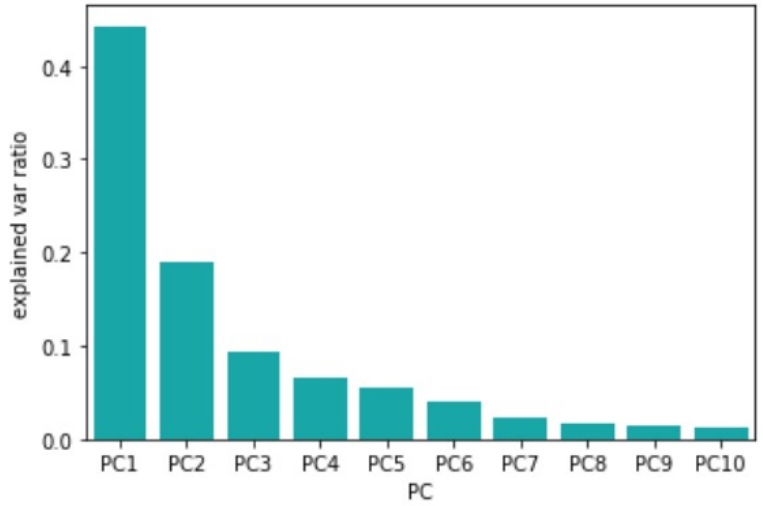
- Plots the eigenvalues from PCA
- Use the pca.explained_variance_ output variable from PCA
- Elbow heuristic: find -45 degree tangent
- Kaiser heuristic: take top-k components with eigenvalue > 1.0

```
df = pd.DataFrame({'eigenvalue': np.sqrt(pca.explained_variance_),
                   'PC':['01','02','03','04','05','06','07','08','09','10']})
sn.lineplot(x = 'PC',y = "eigenvalue", data = df, color="c");
```



Variance ratio plot

```
df = pd.DataFrame({'explained var ratio':pca.explained_variance_ratio_,
                   'PC':['PC1','PC2','PC3','PC4','PC5','PC6','PC7','PC8','PC9','PC10']})
sn.barplot(x = 'PC',y = "explained var ratio", data = df, color="c");
```



Cumulative sum
of explained variance

0	0.442720
1	0.632432
2	0.726364
3	0.792385
4	0.847343
5	0.887588
6	0.910095
7	0.925983
8	0.939879
9	0.951569

Kevyn Collins-Thompson

kevynct@umich.edu

School of Information

Singular Value Decomposition (SVD)

Kevyn Collins-Thompson

Associate Professor of Information and Computer Science
School of Information, University of Michigan

Matrix factorization is everywhere in machine learning

- Topic models/PCA are intimately related to matrix factorization:
  - SVD, PCA, Latent Semantic Indexing, Latent Dirichlet Allocation, Non-Neg Matrix Factorization, ...
  - With different objective functions and constraints
- Matrix factorization connects:
  - Filtering/sensing problems
  - Clustering
  - Topic models
- Our first matrix factorization:
  - Singular value decomposition (SVD)

Brief review of important linear algebra concepts & notation

X Original data matrix

X^* Centered matrix

$\hat{X}$ Centered, normalized matrix

Matrix transpose of X is X^T (but sometimes X’ or X* or Xh, sorry)

Covariance matrix $X^{*T}X^*$

Correlation matrix $\hat{X}^T\hat{X}$

Cross-product matrix X^TX

Orthogonal matrix *U*

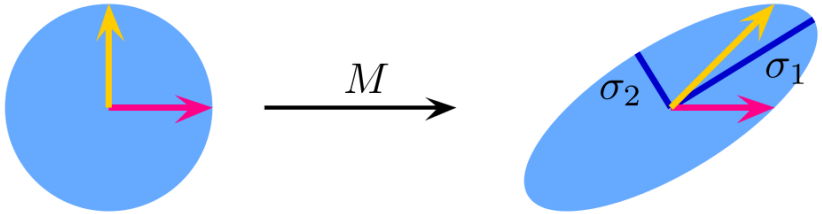
- Rows and columns are orthogonal unit vectors
- Linear ‘unitary’ transformation
- Preserves the inner product of vectors:
  $\langle x, y \rangle = \langle Ux, Uy \rangle$
- Isometry action in Euclidean space
- Rigid rotation, reflection, rotoreflection
- $U^T U = I$ or equivalently $U^T = U^{-1}$

$$\begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix}$$

Rotation by 90 deg cw

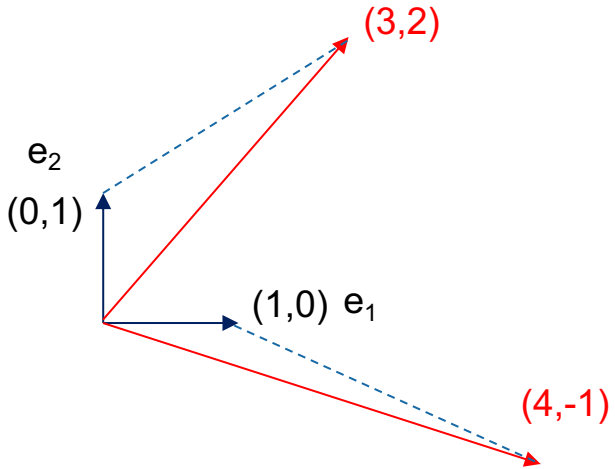
$$\begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Any matrix M can be seen as a linear transformation that acts on a vector *v* when multiplied via dot product: *M v*



$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y \\ a_{21}x + a_{22}y \end{bmatrix}$$

$$\begin{bmatrix} 4 & 3 \\ -1 & 2 \end{bmatrix} \quad \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4x + 3y \\ -x + 2y \end{bmatrix}$$



$$\begin{bmatrix} 4 & 3 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix} \quad \begin{bmatrix} 4 & 3 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 4 \\ -1 \end{bmatrix}$$

Singular Value Decomposition

Any n x p matrix M can be factored into three parts:

$$M = U \Sigma V^{T}$$

U is an *n* x *r* matrix

Σ is an r x r positive diagonal matrix (singular values)

V is a *p* x *r* matrix

r is the rank of M (never > than n or p)

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix}$$

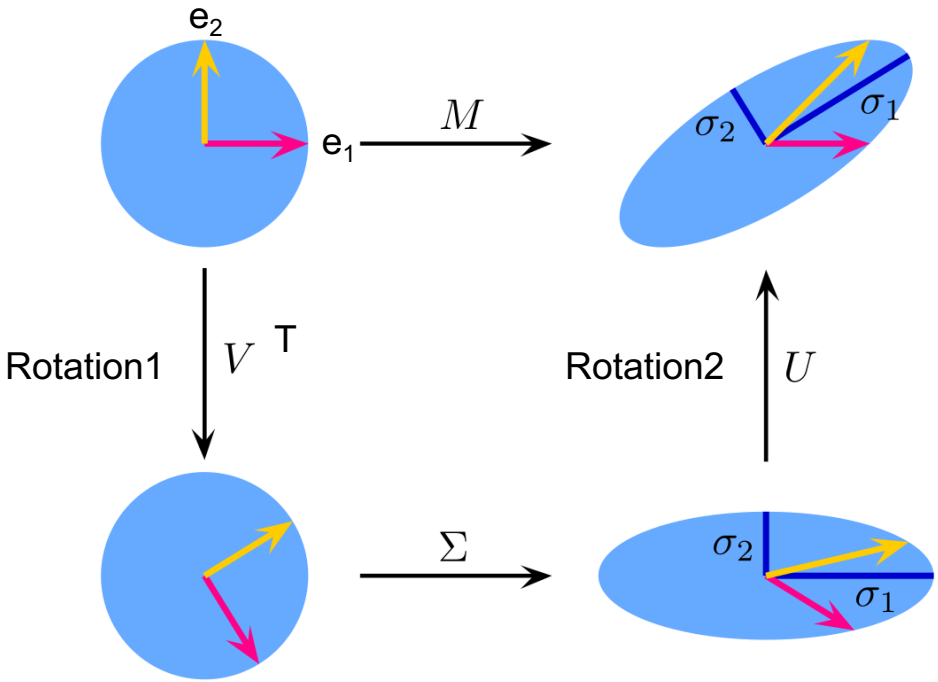
Example r=2

Sometimes the diagonal matrix Σ is written using ***s*** or **D**.

Sometimes for the input matrix we'll use *X* instead of *M.*

The intuitive view of SVD: Any linear transformation *M* can always be decomposed into 3 steps!

Rotation1 (V^T) → Scaling (diagonal matrix Σ) → Rotation2 (U)



$$M = U \cdot \Sigma \cdot V^T$$

Singular value decomposition **UΣV*** of a real 2×2 matrix **M**.

Top: The action of **M**, indicated by its effect on the unit disc *D* and the two canonical unit vectors e_1 and e_2.

Left: The action of $\mathbf{V}^T$, a rotation, on *D*, e_1, and e_2.

Bottom: The action of **Σ**, a scaling by the singular values σ_1 horizontally and σ_2 vertically.

Right: The action of **U**, another rotation.

Adapted from: https://en.wikipedia.org/wiki/Singular_value_decomposition

Don’t believe me? Let’s apply SVD to a simple 2x2 matrix!

$$M = \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix}$$

(As a linear transformation, this matrix takes an input vector and rotates it 90 deg clockwise)

$$SVD(M) = \underset{U}{\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}} \underset{\Sigma}{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}} \underset{V^T}{\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}}$$

Interpretation: rotation 90 deg cw is exactly equivalent to performing these three rotate+scale+rotate steps:

1. V^T: Rotate 180 deg about y-axis (flip x sign)
2. Σ : Scale with identity (no change)
3. U: Rotate 180 deg about origin (flip x and y)

```
In [13]: rotation_90r = np.matrix([[0, +1],[-1, 0]])
         unit_y = np.array([0, 1])
         np.matmul(rotation_90r, np.transpose(unit_y))

Out[13]: matrix([[1, 0]])

In [16]: from scipy import linalg
         U, s, Vh = linalg.svd(rotation_90r)
         U.shape,  s.shape, Vh.shape
         U, s, Vh

Out[16]: (array([[0., 1.],
                 [1., 0.]]),
          array([1., 1.]),
          array([[-1., -0.],
                 [ 0.,  1.]]))

In [19]: S = np.diag(s)
         np.allclose(rotation_90r, np.dot(U, np.dot(S, Vh)))
         np.dot(U, np.dot(S, Vh))

Out[19]: array([[ 0.,  1.],
                [-1.,  0.]])
```

We can also confirm easily that multiplying U, Σ, V^T gives us back *M.*

There are multiple ways we could factor M.
What makes this factoring special? The properties of U, D, and V.

- U and V are 'rigid rotations' a.k.a. orthonormal matrices:
  - $U^T U = I$ and $V^T V = V V^T = I$
  - The squared elements of each column of U and V sum to 1
  - The inner product (dot product) between each pair of columns in U is 0.
  - The inner product between each pair of columns of V is 0

There are multiple ways we could factor M.
What makes this factoring special? The properties of U, D, and V.

- U and V are 'rigid rotations' a.k.a. orthonormal matrices:
  - $U^T U = I$ and $V^T V = V V^T = I$
  - The squared elements of each column of U and V sum to 1
  - The inner product (dot product) between each pair of columns in U is 0.
  - The inner product between each pair of columns of V is 0

- *D* contains a diagonal matrix with nonnegative, decreasing elts:
  - $D_{1,1} \geq D_{2,2} \ldots \geq D_{p,p} \geq 0$
- The diagonal elements of D are the SINGULAR VALUES
- The columns of U and V are the LEFT and RIGHT SINGULAR VECTORS respectively.

The SVD is fast and easy to compute

- Any matrix X can be quickly decomposed this way
- Unique up to sign flips of columns of U & V
- Main engine under the hood:
  - The QR algorithm… with lots of other variants specialized and optimized for specific subtypes of matrices.
  - QR decomposition can be computed via Gram-Schmidt process (but also other more numerically stable methods)
- This seems very cool!
- But what makes the SVD a key algorithm across all of data science?

Example: Using the SVD to create an optimal* approximation of the original matrix X

- You want to approximate X as best as possible with just one pair of vectors.
- Think of this as extreme compression!
- Solution: Do SVD and use the first columns of U and V, multiplied by the first singular value.
- $X \approx U_1 \Sigma_{11} V_1^T$ is the best* possible rank-1 approximation to X.

* In the least-squares sense.

Let's try it on a 2x2 matrix!

$$M = \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix} \qquad SVD(M) = \underset{U}{\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}} \underset{\Sigma}{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}} \underset{V^T}{\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}}$$

$$M \approx \Sigma_{11} U_1 V_1^T \qquad \Sigma_{11} = 1$$

$$\begin{bmatrix} 0 \\ +1 \end{bmatrix} \begin{bmatrix} -1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ -1 & 0 \end{bmatrix} \approx \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Outer product, $\mathbf{uv}^T$ $= \begin{bmatrix} u_1v_1 & u_1v_2 & \dots & u_1v_n \\ u_2v_1 & u_2v_2 & \dots & u_2v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_mv_1 & u_mv_2 & \dots & u_mv_n \end{bmatrix}$

u: $m \times 1$ column vector
v: $n \times 1$ column vector (i.e. $\mathbf{v}^T$ a row vector)