

# Correlated features

Our goal is to find ways to reduce the dimensionality of feature vectors.

Let's explore correlated features in the notebook on [Correlated features \(Unsupervised Correlated Features.ipynb\)](#).

# Principal Components: An alternate basis for our examples

Given that the features may be correlated

- We saw how changing the basis
- Can express the same examples
- In an alternate basis that is perhaps smaller

Let's formalize the notion of [alternate basis \(Unsupervised.ipynb#Alternate-basis\)](#).

# Principal components: introduction

We have seen how we can express the examples in  $\mathbf{X}$  in two coordinate systems

- The one with "original" features
- An alternate basis with "synthetic features"

Principal Components Analysis is the mechanism that we use

- To discover the new, alternate basis
- To find the feature values of examples, as measured in the alternate basis

Let's visit the [notebook section introducing PCA \(Unsupervised.ipynb#What-is-PCA\)](#).

# PCA: The math

The goal of PCA is to find a way of expressing examples  $\mathbf{X}$

- In a new basis  $V^T$
- With feature values  $\tilde{\mathbf{X}}$

$$\mathbf{X} = \tilde{\mathbf{X}}V^T$$

That is, we decompose  $\mathbf{X}$  into a product

- factorization of  $\mathbf{X}$

Let's go to the [notebook section on Matrix factorization \(Unsupervised.ipynb#PCA-via-Matrix-factorization\)](#) to explore how to factor  $\mathbf{X}$ .

# PCA: reducing the number of dimensions

Thus far

- Both the original basis and the new basis  $V$  have consisted of  $n$  basis vectors
- No information has been lost by the basis transformation

$$\mathbf{X} = \tilde{\mathbf{X}}V^T$$

If we are willing to lose some information

$$\mathbf{X}' \approx \mathbf{X}$$

we can achieve dimensionality reduction

- By an alternate basis  $(V^T)'$  of  $r \leq n$  basis vectors
- With synthetic feature vectors  $\mathbf{X}'$  of length  $r$



That is:  $\tilde{\mathbf{X}}'$  is a *reduced dimension* representation.

Questions to consider

- Which synthetic features to drop
- How many synthetic features to drop/keep

Let's go to the notebook section on [dimensionality reduction](#)  
([Unsupervised.ipynb#Dimensionality-reduction](#)).

# Transforming between original and synthetic features

We have thus far been concerned with the transformation

- From original features  $\mathbf{X}$
- To synthetic features  $\tilde{\mathbf{X}}$

We can also go in the opposite direction: from  $\tilde{\mathbf{X}}$  back to original features  $\mathbf{X}$

Let's go to the [notebook section on inverse transformation \(Unsupervised.ipynb#The-inverse-transformation\)](#).

# PCA in action

An example will hopefully tie together all the concepts.

Let's visit the [notebook section on PCA of small digits \(Unsupervised.ipynb#Example:-Reconstructing- \$x\$ -from- \$\tilde{x}\$ -and-the-principal-components\)](#).

# Choosing the number of reduced dimensions

Let's visit the [notebook section on PCA of MNIST \(Unsupervised.ipynb#MNIST-example\)](#) in order to see how the quality of approximation varies with the number of features in  $\tilde{\mathbf{X}}$

# PCA in Finance

Long before Machine Learning became popular, PCA was used to "explain" the yield curve.

A Yield Curve is a vector of features

- Whose length  $n$  corresponds to the number of bond maturities
- $\mathbf{x}_j^{(i)}$  is the yield, on day  $i$  of the  $j^{th}$  bond
  - $j$  increases with maturity

Does the yield of each maturity change (from day to day)

- Independently of other maturities ?
- Or are there a small number of "common factors"/"concepts" that drive daily yield changes ?

PCA can help us answer the question.

In the process, we are also able to *interpret* the common factors

- Which helps our intuition

Let's visit the [notebook section on PCA of the Yield Curve \(Unsupervised.ipynb#PCA-in-Finance\)](#).

# Pseudo Matrix Factorization and Recommender Systems

You have no doubt been to a website that

- Has made a recommendation to you
- Based on your personal "features"

*You might also like ...*



How does this work ?

- You are an example  $\mathbf{x}^{(i)}$ , expressed as a vector of features
  - $\mathbf{x}_j^{(i)}$  is your "rating" for product  $j$
- The number  $n$  of products is large
  - You have rated only a small fraction  $n_i < n$
- You have *not* rated product  $j'$ 
  - How can the system recommend  $j'$  to you ?

PCA to the rescue !

- Perform PCA on  $\mathbf{X}$  ( $m$  is number of users;  $n$  is number of products)
- The Principal Components are
  - "Concepts" that identify groups of products

We will

- Re-express your ratings of concrete product  $\mathbf{x}^{(i)}$
- Into strength of concepts  $\tilde{\mathbf{x}}^{(i)}$
- Find other users  $i'$  with similar strength of concepts
$$\tilde{\mathbf{x}}^{(i')} \approx \tilde{\mathbf{x}}^{(i)}$$
- Deduce that you (user  $i$ ) have similar tastes to user  $i'$
- Recommend to you (user  $i$ ) any product  $j$ 
  - where  $\mathbf{x}_j^{i'}$  is high

This is roughly how Netflix recommendations work.

- Products are Movies
- Principal Components ("concepts") turn out to be Movie genres
  - Action, Comedy, Romance, Gender-specific

So if your movie preferences lean to Comedy, Netflix will recommend to you Comedy-type movies

Although this *seems* like a simple application of PCA

- There is a giant catch !
- $\mathbf{X}$  is sparse (lots of empty entries)
  - How many of the thousands of movies in Netflix have *you* rated ?

How do we factor a matrix with undefined entries ?

Let's go to the [notebook section on Pseudo Matrix Factorization \(Unsupervised.ipynb#Recommender-Systems:-Pseudo-SVD\)](#).

# Pseudo Matrix factorization wrap-up

The techniques in Pseudo factorization are a nice bridge between Classical ML and Deep Learning

- An interesting Loss function
- Not amenable to closed form solution (because of missing entries)
- But approximated using our generic optimization tool
  - Gradient Descent

This type of problem is very typical of those that we will encounter in Deep Learning.

Thus, it is a good transition.



In [4]: `print("Done")`

Done