INT 303 BIG DATA ANALYTICS

Lecture8: Dimension Reduction

Pengfei FAN
pengfei.fan@xjtlu.edu.cn

Xi'an Jiaotong-Liverpool University
西交利物浦大學

Lecture Outline: High Dimensionality and PCA

- Interaction Terms and Unique Parameterizations
- Big Data and High Dimensionality
- Principal Components Analysis (PCA)
- PCA for Regression (PCR)
- PCA for Imputation

Interaction Terms and Unique Parameterizations

NYC Taxi vs. Uber

NYC Taxi vs. Uber

We'd like to compare Taxi and Uber rides in NYC (for example, how much the fare costs based on length of trip, time of day, location, etc.).

A public dataset has 1.9 million Taxi and Uber trips. Each trip is described by $p = 23$ useable predictors (and 1 response variable).

```
In [11]: print(nyc_cab_df.shape)
         nyc_cab_df.head()

         (1873671, 30)
```

Out[11]:

	AWND	Base	Day	Dropoff_latitude	Dropoff_longitude	Ehail_fee	Extra	Fare_amount	Lpep_dropoff_datetime	MTA_tax	...	TMIN	Tip_amount	Tolls_amou
0	4.7	B02512	1	NaN	NaN	NaN	NaN	33.863498	2014-04-01 00:24:00	NaN	...	39	NaN	Na
1	4.7	B02512	1	NaN	NaN	NaN	NaN	19.022892	2014-04-01 00:29:00	NaN	...	39	NaN	Na
2	4.7	B02512	1	NaN	NaN	NaN	NaN	25.498981	2014-04-01 00:34:00	NaN	...	39	NaN	Na
3	4.7	B02512	1	NaN	NaN	NaN	NaN	28.024628	2014-04-01 00:39:00	NaN	...	39	NaN	Na
4	4.7	B02512	1	NaN	NaN	NaN	NaN	12.083589	2014-04-01 00:40:00	NaN	...	39	NaN	Na

5 rows × 30 columns

Interaction Terms: A Review

Recall that an interaction term between predictors X_1 and X_2 can be incorporated into a regression model by including the multiplicative (i.e. cross) term in the model, for example:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 (X_1 * X_2) + \varepsilon$$

Suppose X_1 is a binary predictor indicating whether a NYC ride pickup is a taxi or an Uber, X_2 the length of the trip, and Y is the fare for the ride.

What is the interpretation of β_3?

Including Interaction Terms in Models

Recall that to avoid overfitting, we sometimes elect to exclude a number of terms in a linear model.

It is standard practice to always include the ***main effects*** in the model. That is, we always include the terms involving only one predictor, $\beta_1 X_1, \beta_2 X_2$, etc.

How Many Interaction Terms?

This NYC taxi and Uber dataset has 1.9million Taxi and Uber trips. Each trip is described by $p = 23$ useable predictors (and 1response variable). How many interaction terms are there?

- Two-way interactions: $\binom{p}{2} = \frac{p(p-1)}{2} = 253$
- Three-way interactions: $\binom{p}{3} = \frac{p(p-1)(p-2)}{6} = 1771$
- Etc.

The total number of all possible interaction terms (including main effects) is.

$$\sum_{k=0}^{p} \binom{p}{k} = 2^p \approx 8.3\text{million}$$

What are some problems with building a model that includes all possible interaction terms?

How Many Interaction Terms?

In order to wrangle a data set with roughly 2 million observations, we could use random samples of 100k observations from the dataset to build our models. If we include all possible interaction terms, our model will have 8.3 mil parameters. **We will not be able to uniquely determine 3.mil parameters with only 100k observations**. In this case, we call the model ***unidentifiable***.

To handle this in practice, we can:

- Increase the number of observation
- Consider only scientifically important interaction terms
- Use an appropriate method that account for this issue
- Perform another ***dimensionality reduction*** technique like PCA

Big Data and High Dimensionality

What is 'Big Data'?

In the world of Data Science, the term *Big Data* gets thrown around a lot. What does *Big Data* mean?

A rectangular data set has two dimensions: number of observations (n) and the number of predictors (p). Both can play a part in defining a problem as a *Big Data* problem.

What are some issues when:

- n is big (and p is small to moderate)?
- p is big (and n is small to moderate)?
- n and p are both big?

When *n* is big

When the sample size is large, this is typically not much of an issue from the statistical perspective, just one from the computational perspective.

- Algorithms can take forever to finish. Estimating the coefficients of a regression model, especially one that does not have closed form (like LASSO), can take a while. Wait until we get to Neural Nets!
- If you are tuning a parameter or choosing between models (using CV), this exacerbates the problem.

What can we do to fix this computational issue?

- Perform 'preliminary' steps (model selection, tuning, etc.) on a subset of the training data set. 10% or less can be justified

Keep in mind, big *n* doesn't solve everything

The era of Big Data (aka, large *n*) can help us answer lots of interesting scientific and application-based questions, but it does not fix everything.

Remember the old adage: "**crap in = crap out**". That is to say, if the data are not representative of the population, then modeling results can be terrible. Random sampling ensures representative data.

Xiao-Li Meng does a wonderful job describing the subtleties involved (WARNING: it's a little technical, but digestible): https://www.youtube.com/watch?v=8YLdIDOMEZs

When *p* is big

When the number of predictors is large (in any form: interactions, polynomial terms, etc.), then lots of issues can occur.

- Matrices may not be invertible (issue in OLS).
- Multicollinearity is likely to be present
- Models are susceptible to overfitting

This situation is called *High Dimensionality*, and needs to be accounted for when performing data analysis and modeling.

What techniques have we learned to deal with this?

When Does High Dimensionality Occur?

The problem of high dimensionality can occur when the number of parameters exceeds or is close to the number of observations. This can occur when we consider lots of interaction terms, like in our previous example. But this can also happen when the number of main effects is high.

For example:

- When we are performing polynomial regression with a high degree and a large number of predictors.
- When the predictors are genomic markers (and possible interactions) in a computational biology problem.
- When the predictors are the counts of all English words appearing in a text.

How Does sklearn handle unidentifiability?

In a parametric approach: if we have an over-specified model ($p > n$), the parameters are unidentifiable: we only need $n - 1$ predictors to perfectly predict every observation ($n - 1$ because of the intercept).

So what happens to the 'extra' parameter estimates (the extra β's)?

- the remaining $p - (n - 1)$ predictors' coefficients can be estimated to be anything. Thus there are an infinite number of sets of estimates that will give us identical predictions. There is not one unique set of β's

What would be reasonable ways to handle this situation? How does *sklearn* handle this?

A Framework For Dimensionality Reduction

One way to reduce the dimensions of the feature space is to create a new, smaller set of predictors by taking linear combinations of the original predictors.

We choose $Z_1, Z_2, \ldots, Z_m$, where and where each Z_i is a linear combination of the original p predictors

$$Z_i = \sum_{j=1}^{p} \phi_{ji} X_j$$

for fixed constants ϕ_{ji}. Then we can build a linear regression regression model using the new predictors

$$Y = \beta_0 + \beta_1 Z_1 + \cdots + \beta_m Z_m + \varepsilon$$

Notice that this model has a smaller number ($m+1 < p+1$) of parameters.

A Framework For Dimensionality Reduction (cont.)

A method of dimensionality reduction includes 2 steps:

- Determine an optimal set of new predictors $Z_1, \ldots, Z_m$, for $m < p$.
- Express each observation in the data in terms of these new predictors. The transformed data will have m columns rather than p.

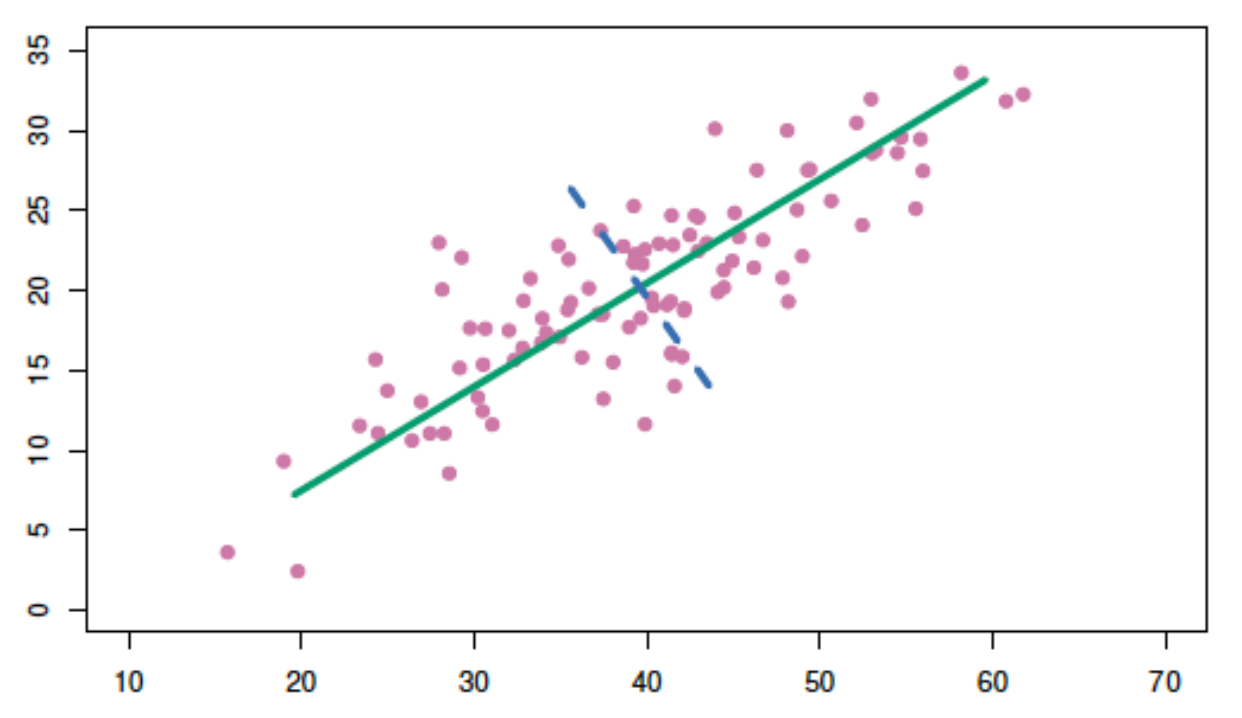
Thereafter, we can fit a model using the new predictors.

The method for determining the set of new predictors (what do we mean by an optimal predictors set?) can differ according to application. We will explore a way to create new predictors that captures the *essential* variations in the observed predictor set.

Principal Components Analysis (PCA)

Principal Components Analysis (PCA)

Principal Components Analysis (PCA) is a method to identify a new set of predictors, as linear combinations of the original ones, that captures the 'maximum amount' of variance in the observed data.



PCA (cont.)

Principal Components Analysis (PCA) produces a list of p **principal components** $Z_1, \ldots, Z_p$ such that

- Each Z_i is a linear combination of the original predictors, and it's vector norm is 1
- The Z_i's are pairwise orthogonal
- The Z_i's are ordered in decreasing order in the amount of captured observed variance.

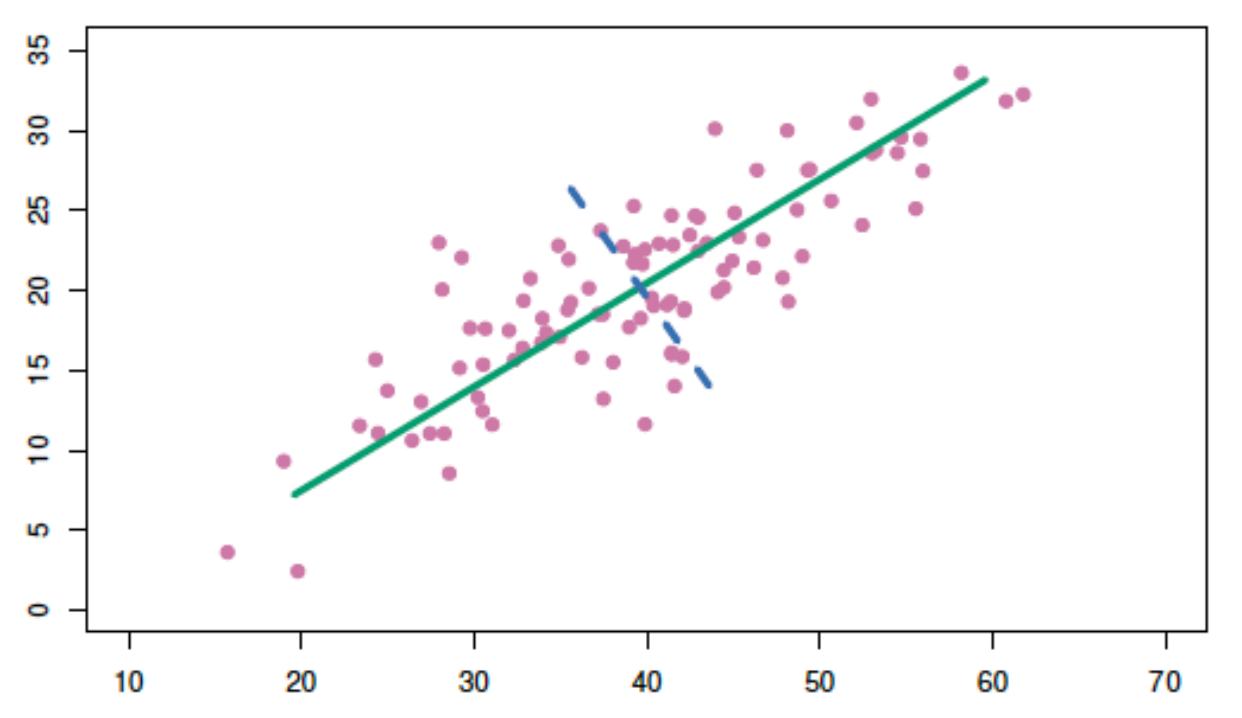
That is, the observed data shows more variance in the direction of Z_1 than in the direction of Z_2.

To perform dimensionality reduction we select the top m principle components of PCA as our new predictors and express our observed data in terms of these predictors.

The Intuition Behind PCA

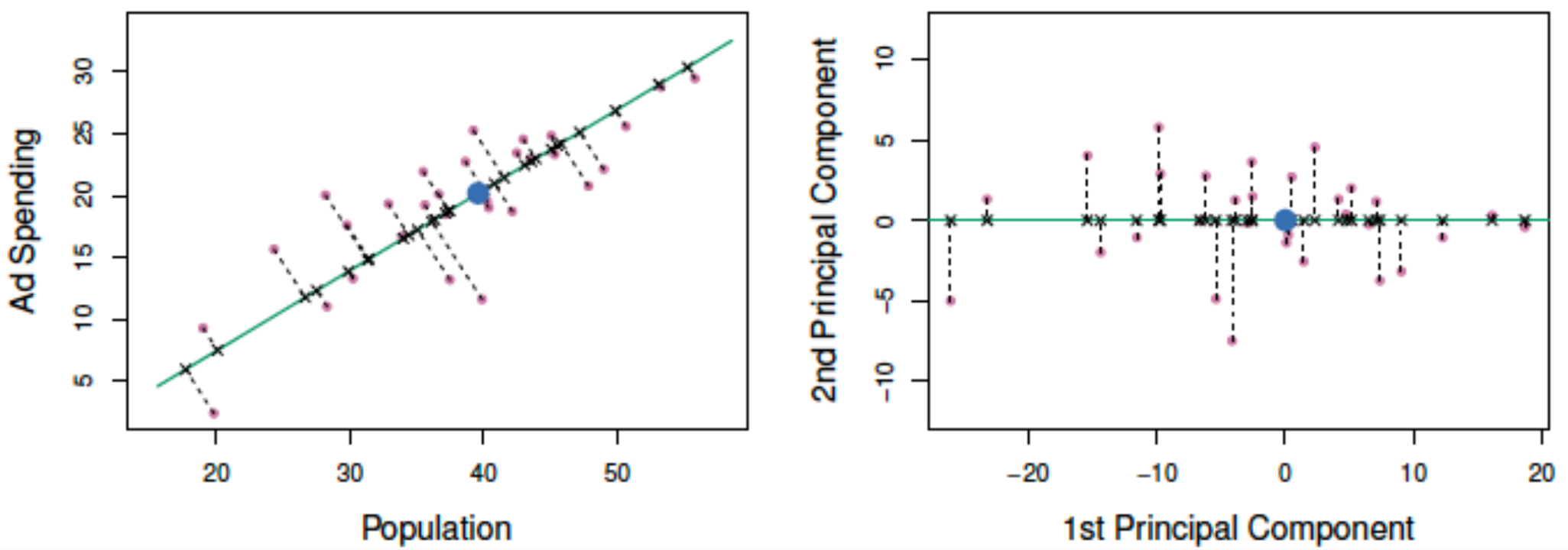
Top PCA components capture the most of amount of variation (interesting features) of the data.

Each component is a linear combination of the original predictors - we visualize them as vectors in the feature space.



The Intuition Behind PCA (cont.)

Transforming our observed data means projecting our dataset onto the space defined by the top *m* PCA components, these components are our new predictors.



The Math behind PCA

PCA is a well-known result from linear algebra. Let **Z** be the n x p matrix consisting of columns $Z_1,\ldots,Z_p$ (the resulting PCA vectors), **X** be the n x p matrix of $X_1,\ldots,X_p$ of the original data variables (each standardized to have mean zero and variance one, and without the intercept), and let **W** be the p x p matrix whose columns are the eigenvectors of the square matrix $\mathbf{X}^T\mathbf{X}$, then:

$$\mathbf{Z}_{n\times p} = \mathbf{X}_{n\times p}\mathbf{W}_{p\times p}$$

Implementation of PCA using linear algebra

To implement PCA yourself using this linear algebra result, you can perform the following steps:

- Standardize each of your predictors (so they each have mean = 0, var = 1).
- Calculate the eigenvectors of the $\mathbf{X}^T\mathbf{X}$ matrix and create the matrix with those columns, **W**, in order from largest to smallest eigenvalue.
- Use matrix multiplication to determine $\mathbf{Z} = \mathbf{XW}$.

Note: this is not efficient from a computational perspective. This can be sped up using Cholesky decomposition.

Let our data matrix X be the score of three students :

Student	Math	English	Art
1	90	60	90
2	90	90	30
3	60	60	60
4	60	60	90
5	30	30	30

Compute the mean of every dimension of the whole dataset

$$\mathbf{A} = \begin{bmatrix} 90 & 60 & 90 \\ 90 & 90 & 30 \\ 60 & 60 & 60 \\ 60 & 60 & 90 \\ 30 & 30 & 30 \end{bmatrix}$$

Matrix A

So, The mean of matrix **A** would be

$$\overline{\mathbf{A}} = [\,66 \quad 60 \quad 60\,]$$

Mean of Matrix A

Compute the *covariance matrix*

$$cov(X,Y) = \frac{1}{n-1}\sum_{i=1}^{n}(Xi - \bar{x})(Yi - \bar{y})$$

	Math	English	Arts
1	90	60	90
2	90	90	30
3	60	60	60
4	60	60	90
5	30	30	30

Matrix A

	Math	English	Art
Math	504	360	180
English	360	360	0
Art	180	0	720

Covariance Matrix of A

Compute Eigenvectors and corresponding Eigenvalues

> *Let* $\mathbf{A}$ *be a square matrix,* $\mathbf{v}$ *a vector and* λ *a scalar that satisfies* $\mathbf{A}v = \lambda v$*, then* λ *is called eigenvalue associated with eigenvector* $\mathbf{v}$ *of* $\mathbf{A}$*.*

$$\det(A-\lambda I) = 0$$

$$\det\left(\begin{pmatrix} 504 & 360 & 180 \\ 360 & 360 & 0 \\ 180 & 0 & 720 \end{pmatrix} - \lambda \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\right)$$

$$\det\begin{pmatrix} 504-\lambda & 360 & 180 \\ 360 & 360-\lambda & 0 \\ 180 & 0 & 720-\lambda \end{pmatrix}$$

$$-\lambda^3 + 1584\lambda^2 - 641520\lambda + 25660800 = 0$$

$$\lambda \approx 44.81966\ldots, \lambda \approx 629.11039\ldots, \lambda \approx 910.06995\ldots$$

Eigenvalues

Eigenvectors

$$\begin{pmatrix} -3.75100\ldots \\ 4.28441\ldots \\ 1 \end{pmatrix}, \begin{pmatrix} -0.50494\ldots \\ -0.67548\ldots \\ 1 \end{pmatrix}, \begin{pmatrix} 1.05594\ldots \\ 0.69108\ldots \\ 1 \end{pmatrix}$$

Eigenvalues

$$\lambda \approx 44.81966\ldots, \lambda \approx 629.11039\ldots, \lambda \approx 910.06995\ldots$$

Eigenvalues

Sort the eigenvectors by decreasing eigenvalues

So, after sorting the eigenvalues in decreasing order, we have

$$\begin{pmatrix} 910.06995 \\ 629.11039 \\ 44.81966 \end{pmatrix}$$

So, *eigenvectors* corresponding to two maximum eigenvalues are :

$$\mathbf{W} = \begin{bmatrix} 1.05594 & -0.50494 \\ 0.69108 & -0.67548 \\ 1 & 1 \end{bmatrix}$$

PCA example in sklearn

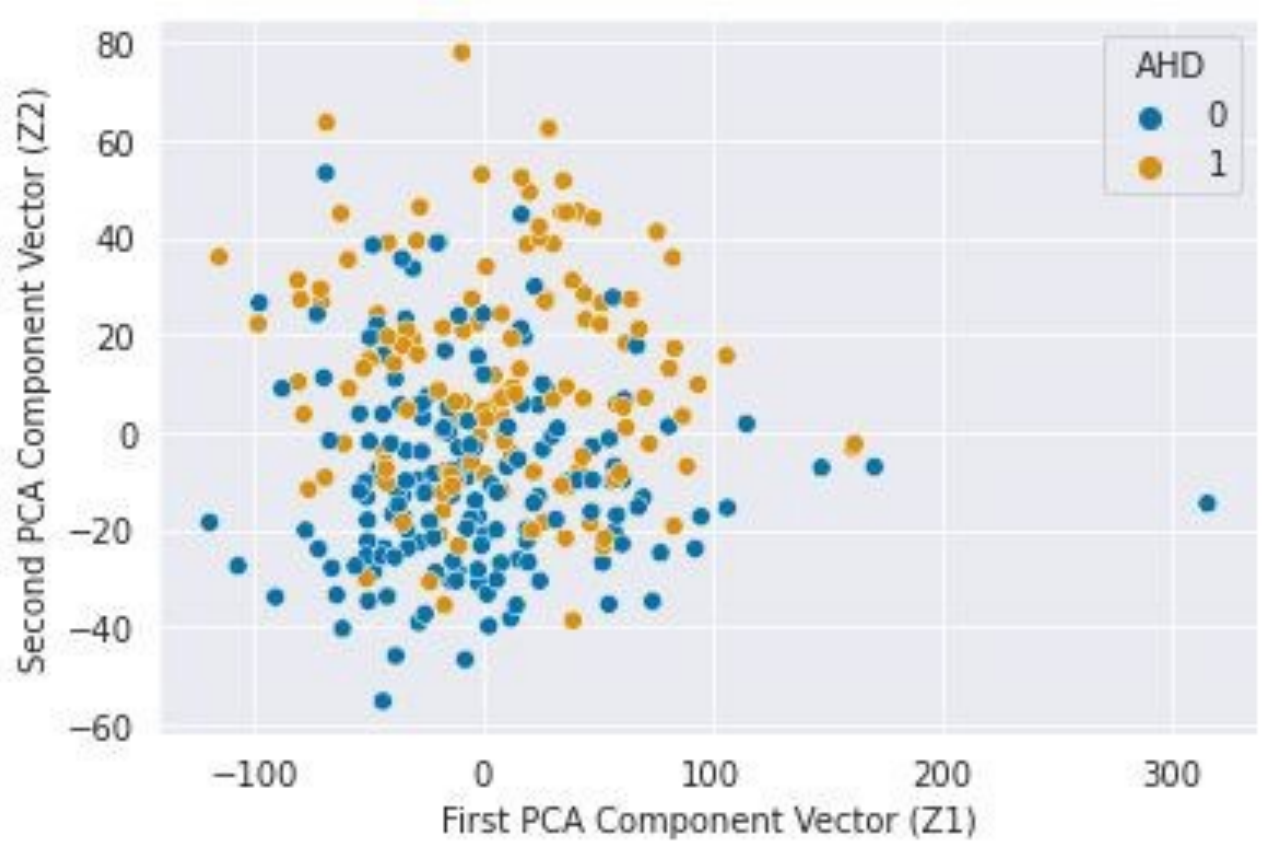
In [11]:
```
X = heart_df[['Age','RestBP','Chol','MaxHR']]

# create/fit the 'full' pca transformation
pca = PCA().fit(X)

# apply the pca transformation to the full predictor set
pcaX = pca.transform(X)

# convert to a data frame
pcaX_df = pd.DataFrame(pcaX, columns=[['PCA1' , 'PCA2', 'PCA3', 'PCA4']])

# here are the weighting (eigen-vectors) of the variables (first 2 at least)
print("First PCA Component (w1):",pca.components_[0,:])
print("Second PCA Component (w2):",pca.components_[1,:])

# here is the variance explained:
print("Variance explained by each component:",pca.explained_variance_ratio_)
```

```
First PCA Component (w1): [ 0.03839966  0.05046168  0.99798051 -0.0037393 ]
Second PCA Component (w2): [ 0.180616    0.10481151 -0.01591307 -0.9778237 ]
Variance explained by each component: [0.74831735 0.15023974 0.0852975  0.01614541]
```

However, PCA is easy to perform in Python using the *decomposition.PCA* function in the *sklearn* package.

PCA example in sklearn

A common plot is to look at the scatterplot of the first two principal components, shown below for the Heart data:



PCA for Regression (PCR)

PCA for Regression (PCR)

PCA is easy to use in Python, so how do we then use it for regression modeling in a real-life problem?

If we use all p of the new Z_j, then we have not improved the dimensionality. Instead, we select the first M PCA variables, $Z_1,...,Z_M$, to use as predictors in a regression model.

The choice of M is important and can vary from application to application. It depends on various things, like how collinear the predictors are, how truly related they are to the response, etc...

What would be the best way to check for a specified problem?

Cross Validation!!!

A few notes on using PCA

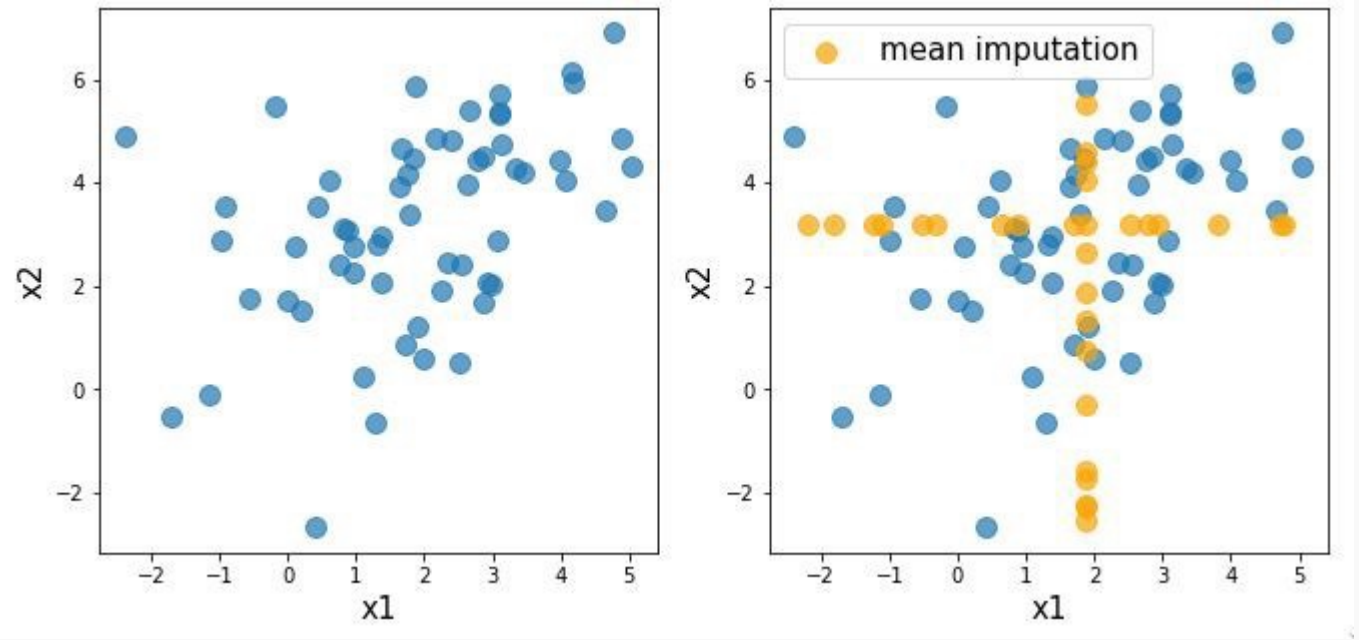
- PCA is an unsupervised algorithm. Meaning? It is done independent of the outcome variable.
    - Note: the component vectors as predictors might not be ordered from best to worst!
- PCA is not so good because:
    1. Direct Interpretation of coefficients in PCR is completely lost. So do not do if interpretation is important.
    2. Will often not improve predictive ability of a model.
- PCA is great for:
    1. Reducing dimensionality in high dimensional settings.
    2. Visualizing how predictive your features can be of your response, especially in the classification setting.
    3. Reducing multicollinearity, and thus may improve the computational time of fitting models.

PCA for Imputation

Naive Imputation Methods

Recall some of the simpler imputation methods we saw last lecture and how they sometimes do more harm than good.

Here we have a positive correlation between two predictors, $x1$ and $x2$. But using mean imputation does not capture this relationship.



What Do We Want from Imputation?

1. We want imputation to take advantage of relationships between predictors, imputing missing values in one predictor using values from the other(s).

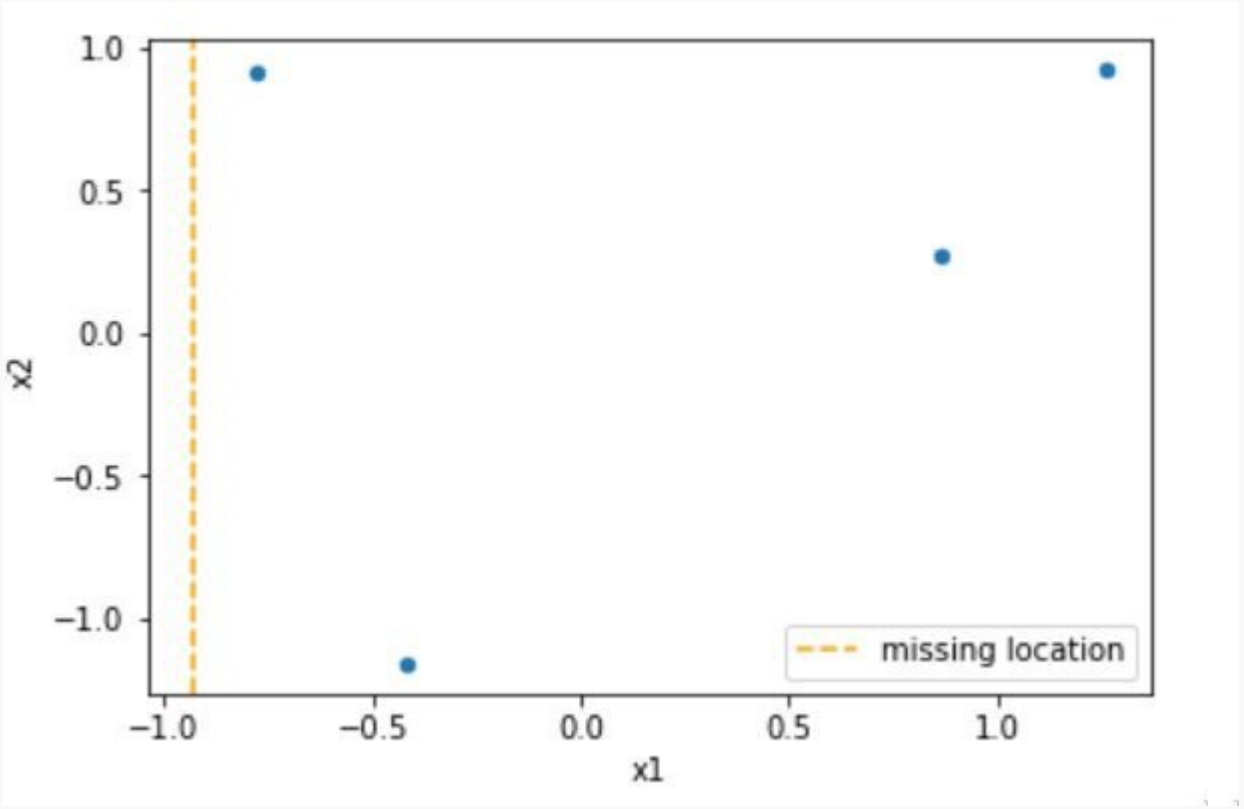
2. When we have many predictors, and observations *i* and *j* have similar values for most them, but *j* is missing predictor *p*, impute using *i*'s value for *p*.

Summary: we want our imputations to take into account (1) **links between variables** and (2) **global similarity between individual observations**. This is the idea behind the **iterative PCA** algorithm for imputation.

Iterative PCA

Here we have a missing x2 value in one of our observations.

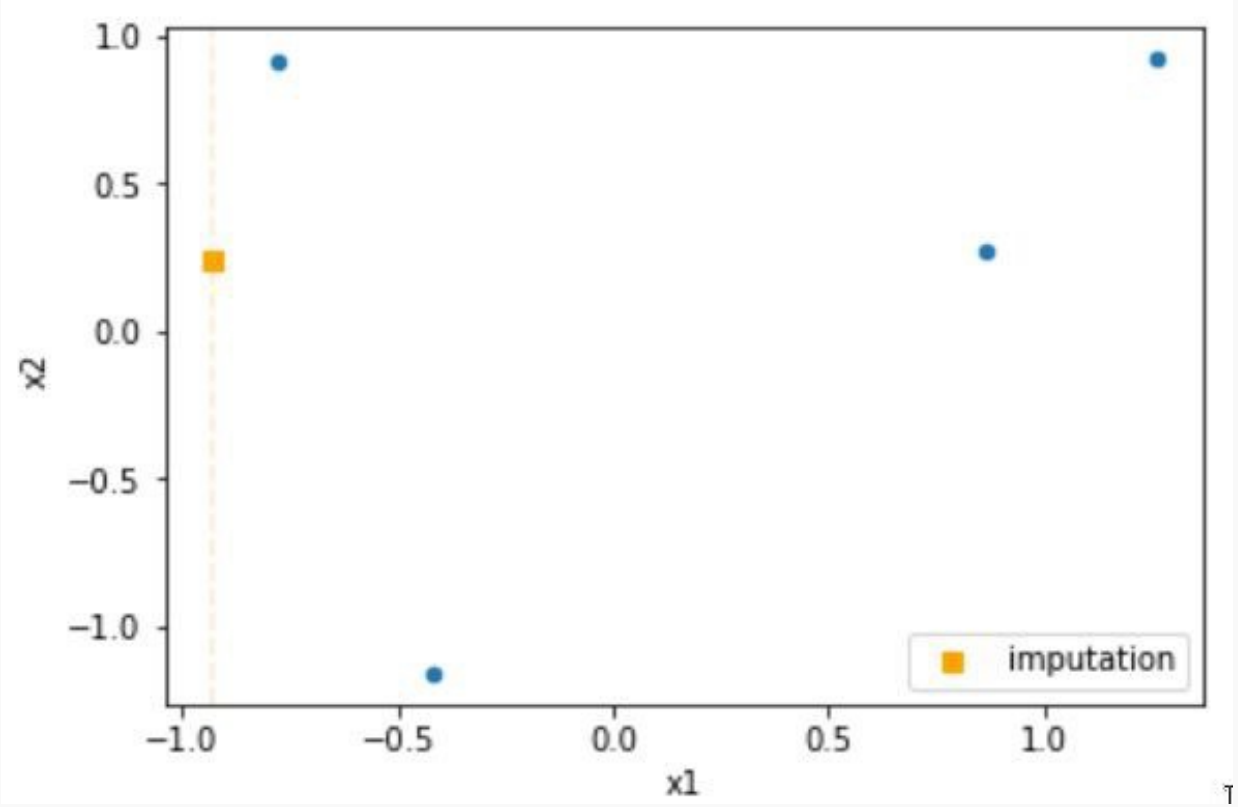
	x1	x2
0	0.865642	0.272073
1	-0.931979	nan
2	1.264153	0.920817
3	-0.781411	0.914296
4	-0.416405	-1.158521



Iterative PCA- Initialization

The first step is to initialize by imputing the mean (though any 'reasonable' value will do).

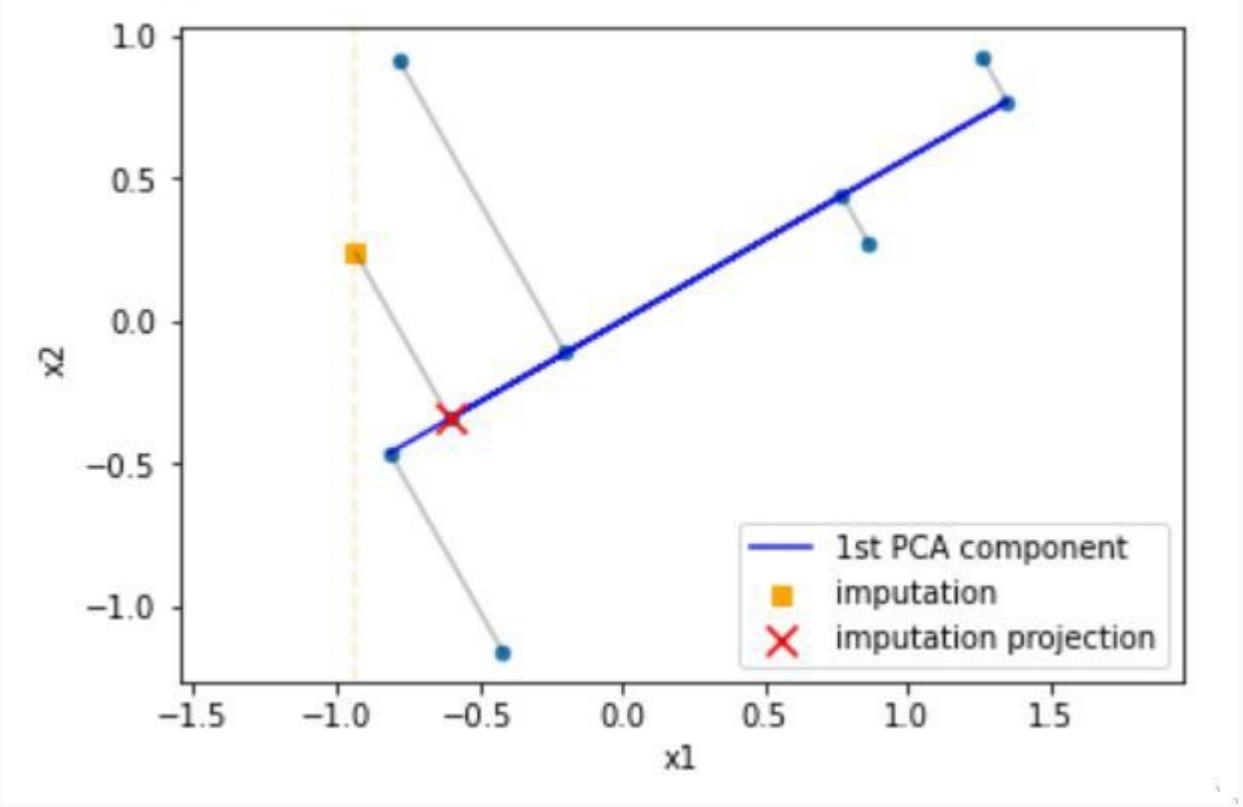
	x1	x2
0	0.865642	0.272073
1	-0.931979	0.237166
2	1.264153	0.920817
3	-0.781411	0.914296
4	-0.416405	-1.158521



Iterative PCA- Projection

Next, we perform PCAon the entire data set and project onto the 1st component.

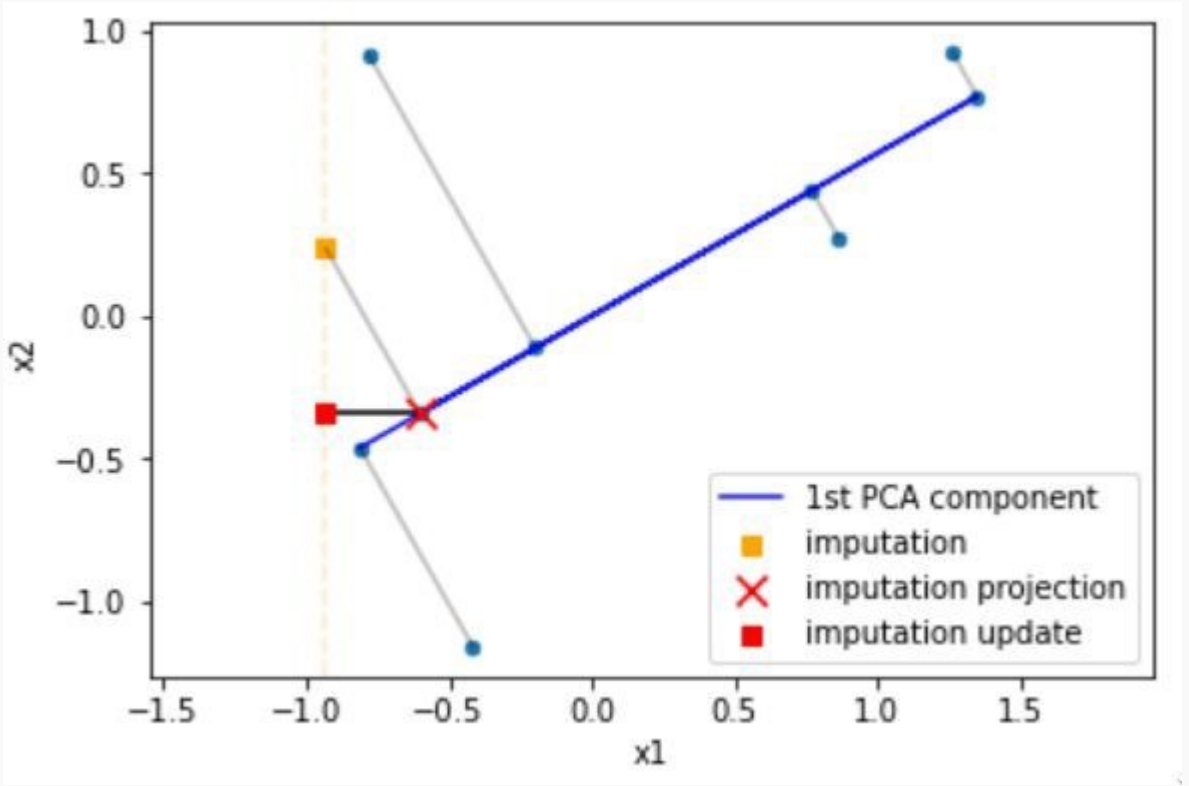
	x1	x2
0	0.865642	0.272073
1	-0.931979	0.237166
2	1.264153	0.920817
3	-0.781411	0.914296
4	-0.416405	-1.158521



Iterative PCA- Update Imputation

The projection's value for the missing predictor is now used to update our imputation.

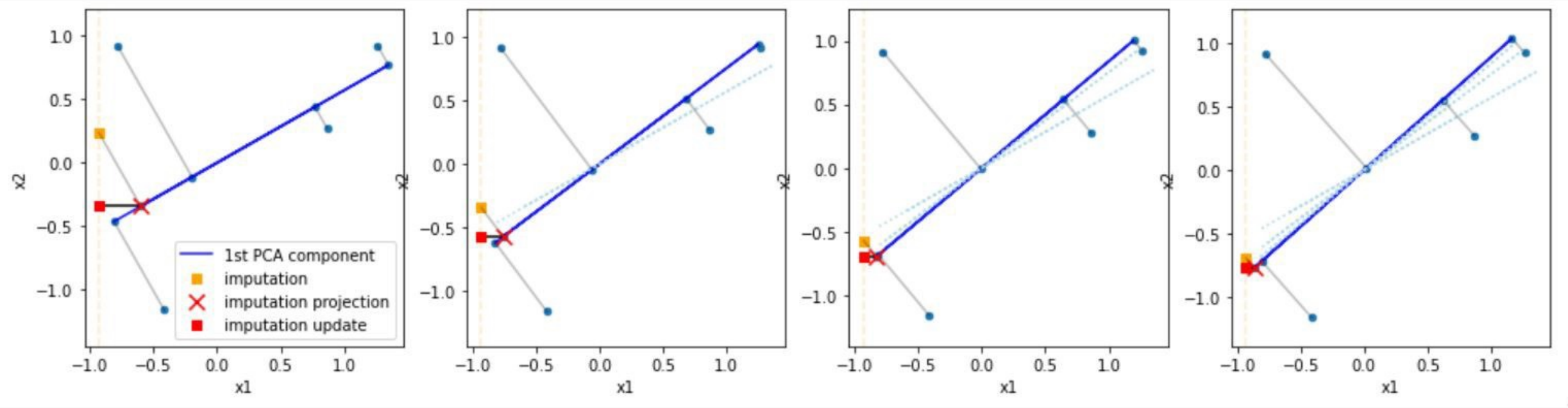
	x1	x2
0	0.865642	0.272073
1	-0.931979	-0.342590
2	1.264153	0.920817
3	-0.781411	0.914296
4	-0.416405	-1.158521



Iterative PCA- Iterate

Notice that the 1st PCA component changes less and less with each iteration.

We simply iterate this process until we converge.



Iterative PCA - Algorithm

1. **Initialize** imputation with reasonable value (e.g., mean)
2. **Iterate** until convergence:
    a. perform **PCA** on the complete data
    b. retain first ***M* components** of PCA (in example $M=1$)
    c. **project** imputation into PCA space
    d. **update** imputation using projection value

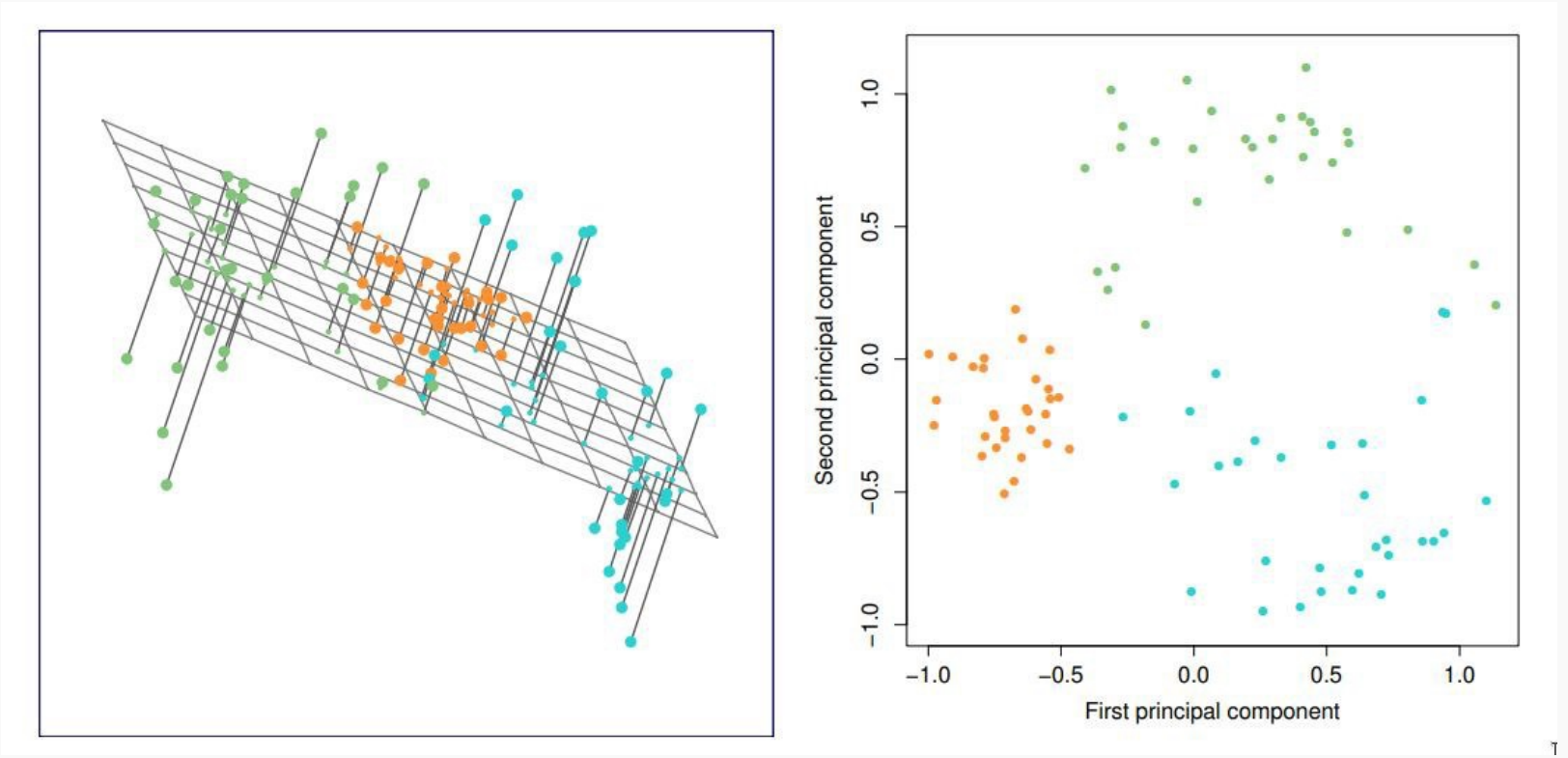
Q: How do we select the number of components to use for our projections?

A: Cross-validation!

In practice this method can overfit, especially in a sparse matrix with many missing values. So often a regularized PCA approach is used which shrinks the singular values of the PCA, making updates less aggressive

An Alternative Interpretation of PCA

- We've seen an interpretation of PCA as finding the directions in the predictor space along which the data varies the most
- An alternative interpretation is that PCA finds a low-dimensional linear surface which is *closest* to the data points



Matrix Completion

Matrix Completion is another application of PCA and is suitable for imputing data which are missing at random. This approach is commonly used in *recommender systems* which deal in very large sparse matrices.

Consider an *n* x *p* matrix of movie ratings by Netflix customers where *n* is the number of customers and *p* is the number of movies. Such a matrix will certainly contain many missing entries.

Imputing these missing values well is equivalent to predicting what customers will think of movies they haven't seen yet.

You can read more about the matrix completion algorithm in the textbook.
(section 12.3 pg. 510)

THANK YOU

VISIT US

WWW.XJTLU.EDU.CN

FOLLOW US

@XJTLU

Xi'an Jiaotong-Liverpool University
西交利物浦大學