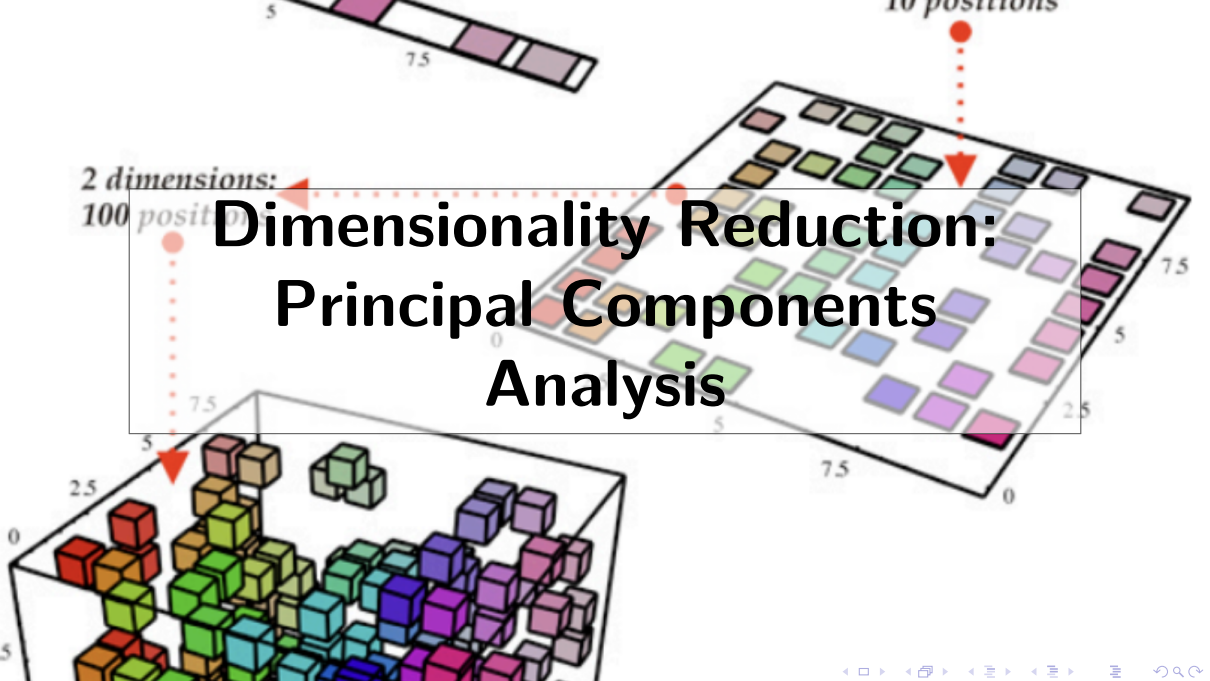


Dimensionality Reduction: Principal Components Analysis

Learning outcomes:

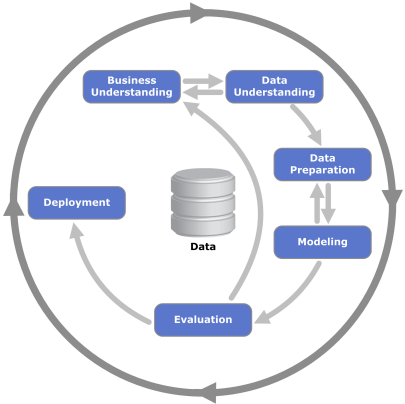
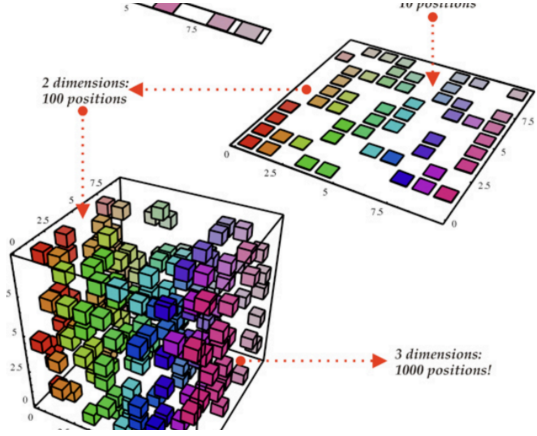


Figure: The CRISP-DM process.

- Summarize the argument for dimensionality reduction;
- Describe the problem PCA is trying to solve;
- Define the PCA algorithm;
- State at least one criteria for deciding how many components to use.

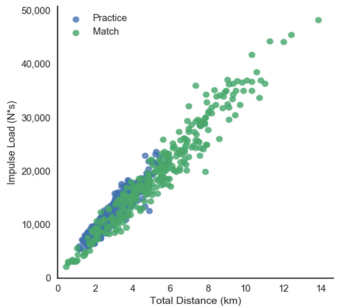
Dimensionality Reduction: what and why?

- In high-dimensional spaces, data points appear to be sparse and not very similar;
- **Curse of dimensionality**: the number of observations needed to cover feature space increases exponentially;
  - Three features with 10 categories has 1000 possibilities;
  - Two features with 10 categories has 100 possibilities;
  - One features with 10 categories has 10 possibilities;
- One way to address this: represent a vector of m features with a vector of d features, $d << m$.



Principal Components Analysis (PCA)

Assumption: the data lie within or close to a much lower dimensional subspace of the high-dimensional space (ex: hyperplane);



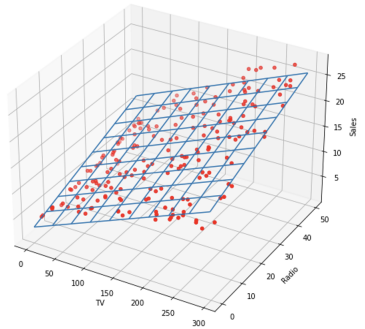
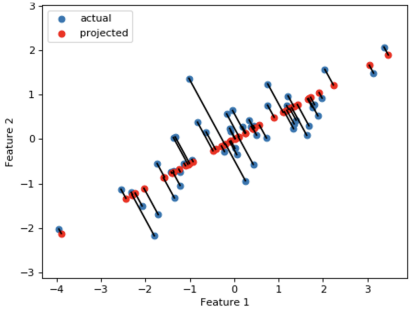


Figure: On the left the data is two dimensional but looks like there is a 1d latent dimension. On the right the data is three dimensional but looks like there is a 2d latent dimension.

Principal Components Analysis (PCA)

Goal of PCA: identify this hyperplane and then project data onto this lower dimensional subspace;



PCA: algorithm

1. Start with a matrix of features X;

PCA: algorithm

1. Start with a matrix of features X;

2. **Standardize** your feature matrix X (for each feature subtract mean and divide by standard deviation) – this will lead to a new matrix X_s;

PCA: algorithm

1. Start with a matrix of features X;
2. **Standardize** your feature matrix X (for each feature subtract mean and divide by standard deviation) – this will lead to a new matrix X_s;
3. Compute the **covariance matrix** for your scaled features:

$$Cov(X_s) = \frac{1}{n-1}(X_s)^T X_s$$

PCA: algorithm

1. Start with a matrix of features X;
2. **Standardize** your feature matrix X (for each feature subtract mean and divide by standard deviation) – this will lead to a new matrix X_s;
3. Compute the **covariance matrix** for your scaled features:

$$Cov(X_s) = \frac{1}{n-1}(X_s)^T X_s$$

4. Solve for the **eigenvectors** and **eigenvalues** of our covariance matrix $Cov(X_s)$;

PCA: algorithm

1. Start with a matrix of features X;
2. **Standardize** your feature matrix X (for each feature subtract mean and divide by standard deviation) – this will lead to a new matrix X_s;
3. Compute the **covariance matrix** for your scaled features:

$$Cov(X_s) = \frac{1}{n-1}(X_s)^T X_s$$

4. Solve for the **eigenvectors** and **eigenvalues** of our covariance matrix $Cov(X_s)$;
5. Multiply our standardized feature matrix X_s by the eigenvectors vectors to project them.

Step 1: start with X

- Example data:
  - One hundred observations;
  - Two numerical variables;
  - $x_2 = 2x_1 - 3 + \varepsilon$;
- x_1 has mean ≈ 0.54 and standard deviation ≈ 0.28;
- x_2 has mean ≈ -1.91 and standard deviation ≈ 0.56;

	x_1	x_2
0	0.873429	-1.237948
1	0.968541	-1.062855
2	0.869195	-1.228109
3	0.530856	-1.900809
4	0.232728	-2.463162
...	...	...
95	0.298625	-2.196156
96	0.138385	-2.620040
97	0.635398	-1.754138
98	0.008308	-3.180289
99	0.788583	-1.455530

100 rows × 2 columns

Step 2: standardize X to make X_s

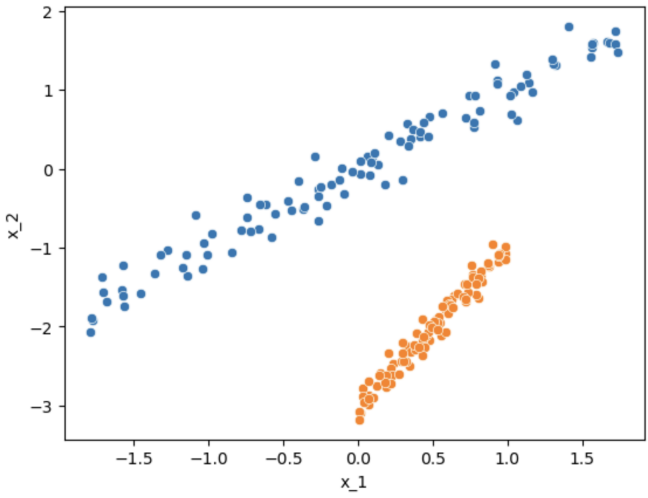
- For each column:
  - Compute the mean and standard deviation;
  - For each entry in the column subtract the mean and divide by the standard deviation;
- After this x_1 and x_2 will have mean 0 and standard deviation 1;
- Easy to do manually, or can use StandardScaler from sklearn.

	x_1	x_2
0	1.322987	1.310675
1	1.665125	1.615516
2	1.307753	1.327807
3	0.090668	0.156623
4	-0.981766	-0.822445
...	...	...
95	-0.744720	-0.357583
96	-1.321143	-1.095573
97	0.466731	0.411980
98	-1.789058	-2.070977
99	1.017775	0.931861

100 rows × 2 columns

Step 2: standardize X to make X_s

The original data X is in orange and the standardized data X_s is in blue.



Step 3: compute the covariance matrix – but what is 'covariance'?!

- Remember **standard deviation**?

Step 3: compute the covariance matrix – but what is 'covariance'?!

- Remember **standard deviation**?
- Variance = square root of standard deviation – describes within feature variation;

Step 3: compute the covariance matrix – but what is 'covariance'?!

- Remember **standard deviation**?
- Variance = square root of standard deviation – describes within feature variation;
- But what if we have multiple features – how do we measure how one feature varies with another? **Covariance** is a measure of between feature variation!

Step 3: compute the covariance matrix – but what is 'covariance'?!

- Remember **standard deviation**?
- Variance = square root of standard deviation – describes within feature variation;
- But what if we have multiple features – how do we measure how one feature varies with another? **Covariance** is a measure of between feature variation!
- But didn't we already have **correlation** to do this job? We did – it is a standardized version of covariance:

$$Cov(x, y) = \frac{\sum_j (x_j - \overline{x})(y_j - \overline{y})}{n - 1} \text{ and } Corr(x, y) = \frac{Cov(x, y)}{\sigma_x \sigma_y}.$$

Step 3: compute the covariance matrix – wait, but why?!

- Remember that we want our features to collectively predict our response variable;
- The amount our feature varies with our response can be thought of as the information encoded in our feature that is able to predict our response;

Step 3: compute the covariance matrix – wait, but why?!

- Remember that we want our features to collectively predict our response variable;
- The amount our feature varies with our response can be thought of as the information encoded in our feature that is able to predict our response;
- Covariance is a way to measure how much information is available in our data across features;
  - ML (and hypothesis testing) is all about separating our observations;
  - The features that describe our observations provide this separation;
- We want to minimize the correlation between our features;
  - If the features contain the same information (covary highly) then adding more won't improve our ability to separate observations;
  - Additionally, we may not know – or be able to clearly solve – for which feature is important for our response.

Step 3: compute the covariance matrix

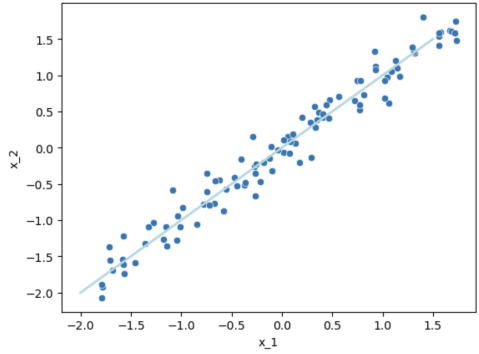
Compute the **covariance matrix** for your scaled features $Cov(X_s) = \frac{1}{n-1}(X_s)^T X_s$:

$$\frac{1}{n-1} \times \overbrace{\begin{pmatrix} 1.322987 & 1.665125 & 1.307753 & \dots \\ 1.310675 & 1.615516 & 1.327807 & \dots \end{pmatrix}}^{X_s^T} \times \overbrace{\begin{pmatrix} 1.322987 & 1.310675 \\ 1.665125 & 1.615516 \\ 1.307753 & 1.327807 \\ 0.090668 & 0.156623 \\ \vdots & \vdots \end{pmatrix}}^{X_s}$$

$$= \begin{pmatrix} 1.0000 & 0.9821 \\ 0.9821 & 1.0000 \end{pmatrix} = Cov(X_s)$$

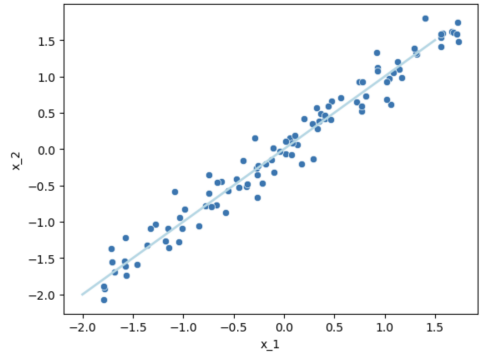
Before we get to step 4...

- Looks like we don't really have two dimensions of information here...
- So, what if we put a line through the point cloud? Looks like this line:
  - Was described by a combination of x_1 and x_2;
  - Went along the axis of highest variance;
- If we had a new feature along this line, we might be able to get away with only using it...



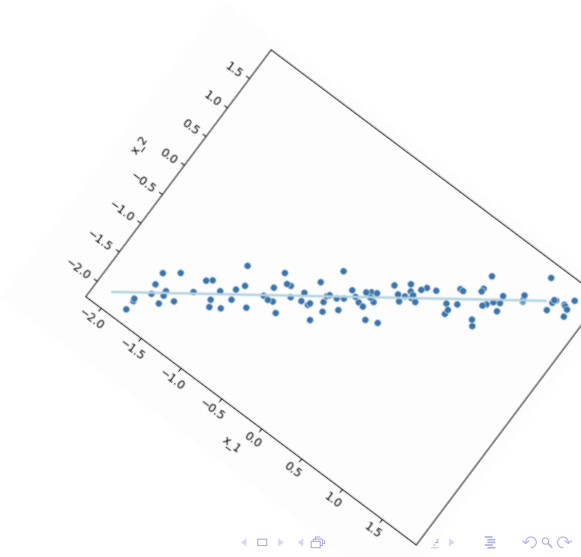
Before we get to step 4...

- We think in **Euclidean space** – the axes are **orthogonal**;
- When we plot features, we represent them on these orthogonal axes;
- The axes on which we operate are **basis vectors**;
- There is nothing stopping us from describing our original features in a new basis...



Before we get to step 4...

- We think in **Euclidean space** – the axes are **orthogonal**;
- When we plot features, we represent them on these orthogonal axes;
- The axes on which we operate are **basis vectors**;
- There is nothing stopping us from describing our original features in a new basis... what if we rotate the space to make new axes?



Step 4: Solve for the eigenvectors and eigenvalues of $Cov(X_s)$ – what?

- If we have a matrix M then the Eigenvalues and Eigenvectors will satisfy:

$$MU = \lambda U$$

  - λ is the **Eigenvalue**;
  - U is the **Eigenvector**;

Step 4: Solve for the eigenvectors and eigenvalues of $Cov(X_s)$ – what?

- If we have a matrix M then the Eigenvalues and Eigenvectors will satisfy:

$$MU = \lambda U$$

  - λ is the **Eigenvalue**;
  - U is the **Eigenvector**;

- Properties of Eigenvalues and Eigenvectors:
  1. Each Eigenvalue has an associated Eigenvector;
  2. Each pair of Eigenvectors for a symmetric matrix are orthogonal;
  3. Any symmetric matrix M is a sum of Eigenvectors weighted by Eigenvalues:

$$M = \sum_i \lambda_i U_i U_i^T.$$

Step 4: Solve for the eigenvectors and eigenvalues of $Cov(X_s)$ – why?

- We can represent all the information in the covariance matrix with a bunch of orthogonal vectors (the Eigenvectors):

$$Cov(X_s) = \sum_i \lambda_i U_i U_i^T;$$

  - The Eigenvectors are a new basis (new axes) in which feature variance is maximized along the each dimension;
  - The Eigenvalues are the importance (explained variance) of each axis;

Our covariance matrix:

$$Cov(X_s) = \begin{pmatrix} 1.0000 & 0.9821 \\ 0.9821 & 1.0000 \end{pmatrix}$$

has Eigenvalues/vectors of:

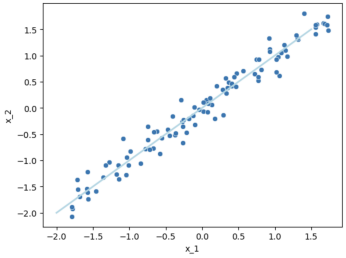
$$\lambda_1 = 1.9821, U_1 = \begin{pmatrix} 0.7071 \\ 0.7071 \end{pmatrix}$$

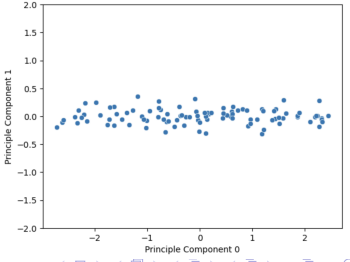
$$\lambda_2 = 0.0179, U_2 = \begin{pmatrix} -0.7071 \\ 0.7071 \end{pmatrix}$$

Step 5: Multiply X_s by the eigenvectors vectors to project

$$\overbrace{\begin{pmatrix} 1.322987 & 1.310675 \\ 1.665125 & 1.615516 \\ 1.307753 & 1.327807 \\ 0.090668 & 0.156623 \\ \vdots & \vdots \end{pmatrix}}^{X_s} \times \overbrace{\begin{pmatrix} 0.7071 \\ 0.7071 \end{pmatrix}}^{U_1} = \overbrace{\begin{pmatrix} 1.8623 \\ 2.3198 \\ 1.8636 \\ 0.1748 \\ \vdots \end{pmatrix}}^{\text{PC 1}}$$

$$\overbrace{\begin{pmatrix} 1.322987 & 1.310675 \\ 1.665125 & 1.615516 \\ 1.307753 & 1.327807 \\ 0.090668 & 0.156623 \\ \vdots & \vdots \end{pmatrix}}^{X_s} \times \overbrace{\begin{pmatrix} -0.7071 \\ 0.7071 \end{pmatrix}}^{U_2} = \overbrace{\begin{pmatrix} -0.0087 \\ -0.0350 \\ 0.0141 \\ 0.0466 \\ \vdots \end{pmatrix}}^{\text{PC 2}}$$





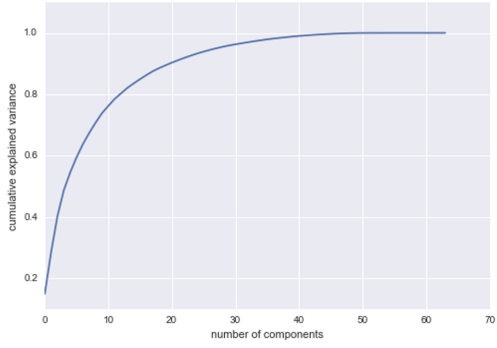
So, how do we use this?

- Now that you have these principal components you can:
  - use them as new features;
  - use them for visualization;
  - use them to compress the data.

- How many components should you use? Compute **cumulative explained variance**:
  - Sort Eigenvalues largest to smallest;
  - Compute:

$$\frac{\sum_{i=1}^{k} \lambda_i}{\sum_i \lambda_i}$$

  - Choose the number of components that explains a large portion of the variance, e.g. 95%.



PCA – summary

- PCA gives us principal components that we can use as new features;
  - Mixtures of our previous features – hard to interpret;
  - Governed by the data that we have (unsupervised);
  - The information is distilled into the basis vectors with the largest eigenvalues;
  - The noise in our data becomes concentrated in the basis vectors with the smallest eigenvalues;
  - We can use fewer features (principal components) to describe the relationships in our data – Dimensionality Reduction;

- IRL we don't perform PCA this way:
  - Covariance matrix will be very large;
  - We use randomized algorithms or SVD to avoid computing the covariance matrix;
  - Good libraries (e.g., scikit-learn) should provide implementations.