IT496: Introduction to Data Mining

Lecture 21

Dimensionality Reduction

(Slides are created from the lecture notes of ML class at UCD, Ireland)

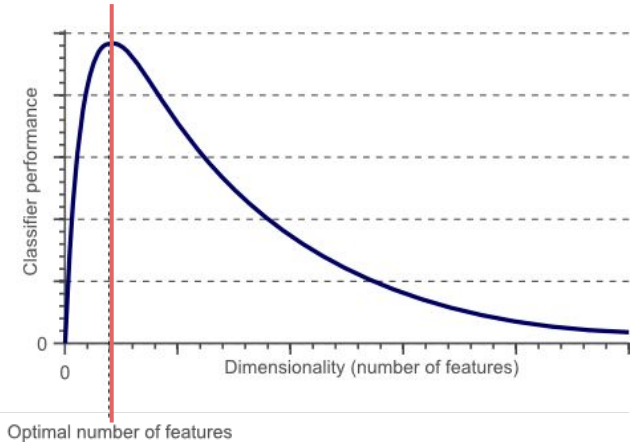
Arpit Rana

6th October 2023

Motivation

Curse of dimensionality is the phenomenon whereby many machine learning algorithms can perform poorly on high-dimensional data.

- Intuitively, adding more features (dimensions) to a dataset should provide more information about each example, making prediction easier.
- In reality, we often reach a point where adding more features no longer helps, or can even reduce predictive power.



In practice, to build a good predictive model, the number of examples required per feature increases exponentially with number of features.

Motivation

There are often other reasons why we might want to reduce the number of features used to represent data:

- less training time and less computational resources
- extremely useful for data visualization
- one way to avoid the problem of overfitting
- takes care of multicollinearity
- filters out some noise and unnecessary details in the data and increases the overall performance

Understanding which features in our data are informative and which are not is an important knowledge discovery task.

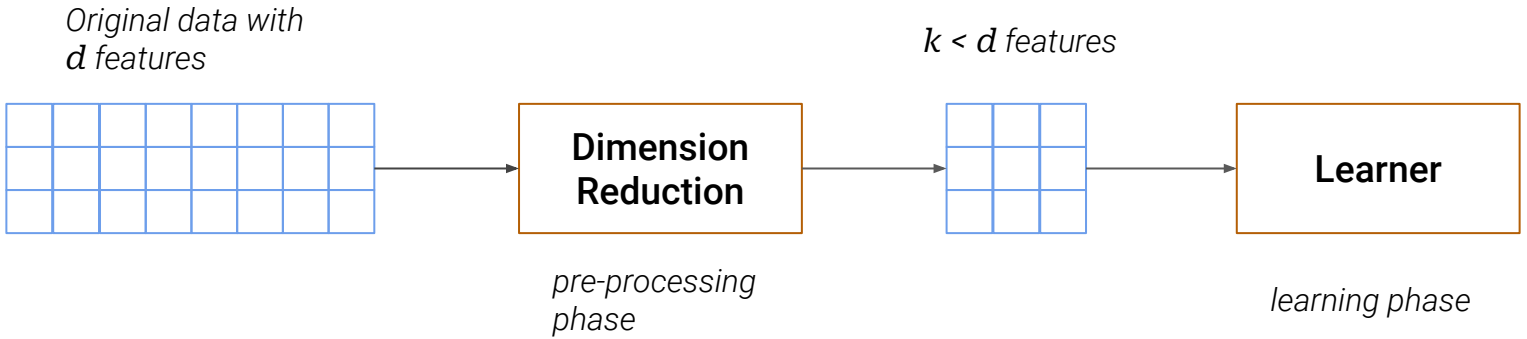
Dimensionality Reduction: Formal Definition

Dimension (or Dimensionality) reduction is -

- the transformation of data from a high-dimensional space into a low-dimensional space
- while preserving as much of the variation in the original dataset as possible.

Dimensionality Reduction

- Basic idea is to try to beat the curse of dimensionality:
  - Apply pre-processing techniques to reduce the number of features used to represent a dataset.
  - Then build a model on the smaller feature set.



- In many (but not all) cases, the additional information that is lost by removing some features is (more than) compensated by higher classifier accuracy in the lower dimensional space.

Dimensionality Reduction: Strategies

Feature Selection

Select subset of the original features, use them to represent the data.

Filters

Rank features using IG, Gini Index independent to the choice of the learner

Wrappers

Select features as per their effect on the performance of the learning algorithm

Feature Extraction (Transformation)

Create a new set of dimensions, use them to represent the data.

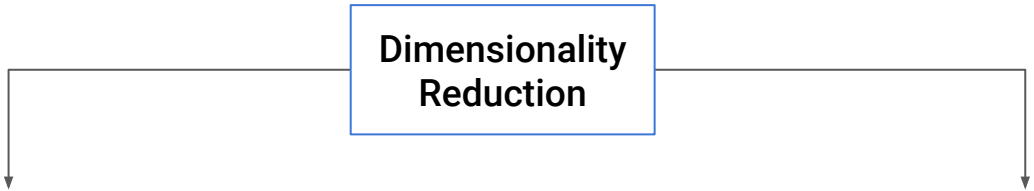
Linear Methods (Projection)

Project training instances to a much lower-dimensional subspace, e.g., PCA, LDA

Non-linear Methods (Manifold Learning)

Unroll the twists and turns in a lower dimensional subspace, e.g., k-PCA

Dimensionality Reduction: Strategies



Feature Selection

Select subset of the original features, use them to represent the data.

Filters

Rank features using IG, Gini Index independent to the choice of the learner

Wrappers

Select features as per their effect on the performance of the learning algorithm

Feature Extraction (Transformation)

Create a new set of dimensions, use them to represent the data.

Linear Methods (Projection)

Project training instances to a much lower-dimensional subspace, e.g., **PCA**, LDA

Non-linear Methods (Manifold Learning)

Unroll the twists and turns in a lower dimensional subspace, e.g., k-PCA

Linear Transformations

In a linear transformation, we map data to new variables, which are linear functions of the original variables.

- **Example**: Bill owes Mary £ 5 and CHF 15 after a holiday.
  - Current exchange rates: € : £ → 1.25 : 1, € : CHF → 0.8:1
- Based on rates, Bill owes € (1.25 x 5 + 0.8 x 15) = € 18.25
- We can view this as a linear transformation…

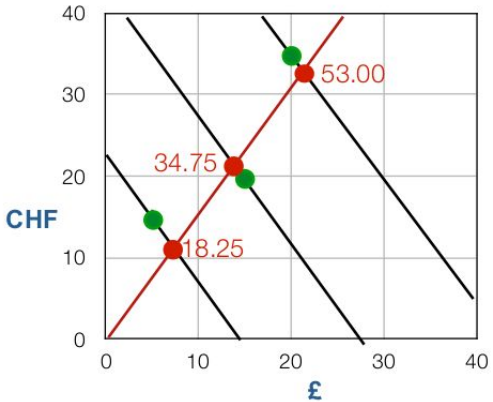
£	CHF
5	15
15	20
20	35

×

1.25
0.8

=

€
18.25
34.75
53.00



Principal Component Analysis (PCA)

Basic Idea

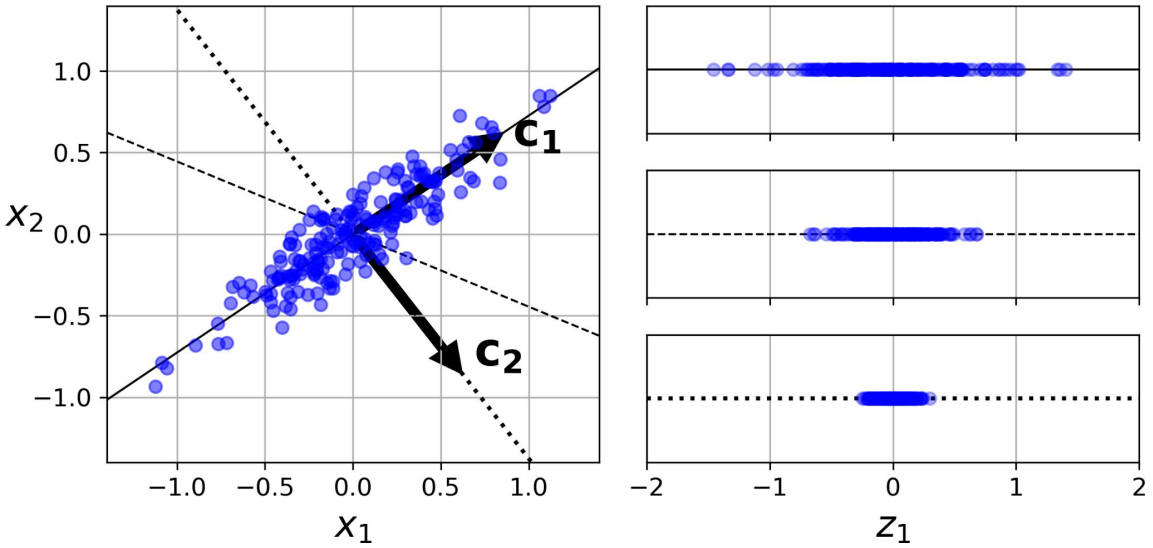
- *Projection methods* map the original d-dimensional space to a new ($k < d$)-dimensional space, with the minimum loss of information.
    - PCA identifies the hyperplane that lies closest to the data, and
    - then it projects the data onto it.

How to choose the hyperplane?

PCA: Preserve the Variance

"Good" spaces for projections are characterised by -

- preserving most of the useful information in the data - the ***variation*** in the data.
- i.e., minimize the mean squared distance between the original dataset and its projection.



This new dimension has more variation - i.e. more information has been preserved.

Principal Components

PCA identifies the axis that accounts for the largest amount of variance in the training set.

- It finds a second axis, orthogonal to the first one, that accounts for the largest amount of *remaining variance*.
- Then, a third axis, orthogonal to both previous axes, and a fourth, a fifth, and so on—as many axes as the number of dimensions in the dataset.

The unit vector that defines the i^{th} axis is called the i^{th} *principal component* (PC).

Principal Components

Singular Value Decomposition (SVD)

- that can decompose the training set matrix **X** into the matrix multiplication of three matrices $\mathbf{U}\,\Sigma\,\mathbf{V}^T$, where **V** contains ***unit vectors*** that define all the principal components.

$$\mathbf{V} = \begin{pmatrix} | & | & & | \\ \mathbf{c_1} & \mathbf{c_2} & \cdots & \mathbf{c_n} \\ | & | & & | \end{pmatrix}$$

Projecting down to d dimension

$$\mathbf{X}_{d\text{-proj}} = \mathbf{X}\mathbf{W}_d$$

```
X_centered = X - X.mean(axis=0)
U, s, Vt = np.linalg.svd(X_centered)
c1 = Vt.T[:, 0]
c2 = Vt.T[:, 1]
```

```
W2 = Vt.T[:, :2]
X2D = X_centered.dot(W2)
```

PCA using Scikit-learn

- Scikit-Learn's PCA class implements PCA using SVD decomposition just like we did before.
- The following code applies PCA to reduce the dimensionality of the dataset down to *two dimensions* (note that it automatically takes care of centering the data):

```
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)
X2D = pca.fit_transform(X)
```

- We can access the principal components using the `components_` variable (note that it contains the PCs as row vectors, i.e., transpose of W_d).
    - The first principal component is equal to `pca.components_.T[:,0]`

Explained Variance Ratio

- Another very useful piece of information is the *explained variance ratio* of each principal component, available via the `explained_variance_ratio_` variable.

- It indicates the proportion of the dataset's variance that lies along the axis of each principal component.

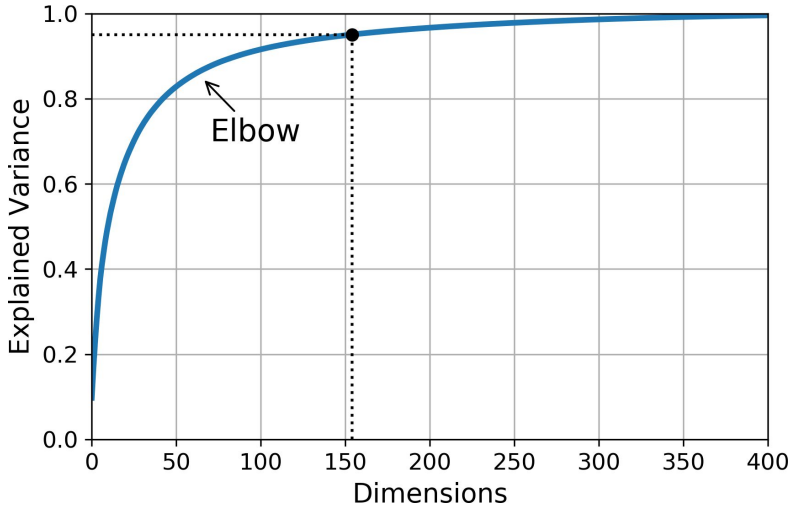
```
>>> pca.explained_variance_ratio_
array([0.84248607, 0.14631839])
```

- This tells us that 84.2% of the dataset's variance lies along the first axis, and 14.6% lies along the second axis.

Choosing the Right Number of Dimensions

It is generally preferable to choose the number of dimensions that add up to a sufficiently large portion of the variance (e.g., 95%).

```
pca = PCA(n_components=0.95)
X_reduced = pca.fit_transform(X_train)
```



PCA for Compression

It is also possible to *decompress* the reduced dataset back to its original dimensions by applying the *inverse transformation* of the PCA projection.

$$\mathbf{X}_{\text{recovered}} = \mathbf{X}_{d\text{-proj}}\mathbf{W}_d^T$$

- This won't give you back the exact original data, since the projection lost a bit of information (e.g., within the 5% variance that was dropped), but it will likely be quite close to the original data.

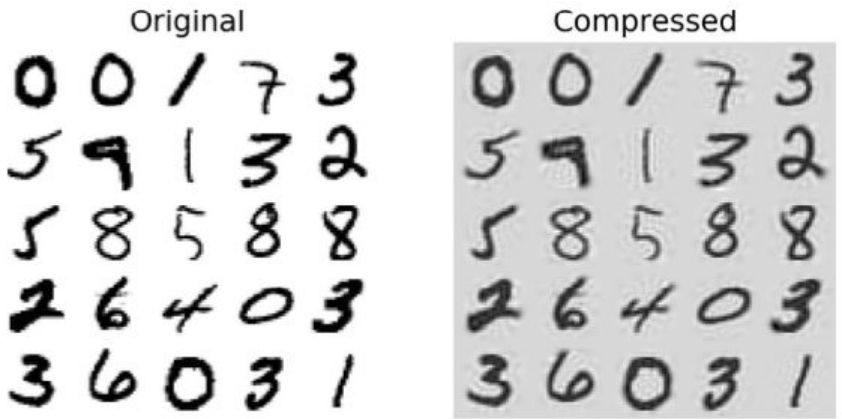
- The mean squared distance between the original data and the reconstructed data (compressed and then decompressed) is called the ***reconstruction error***.

$$Error_{reconstruction} = MSE(X, X_{recovered})$$

PCA for Compression

Applying PCA to the MNIST dataset while preserving 95% of its variance: in place of 784 (28 x 28 shape images) original features, just using 154 (nearly 20% of the original).

```
pca = PCA(n_components = 154)
X_reduced = pca.fit_transform(X_train)
X_recovered = pca.inverse_transform(X_reduced)
```



Conclusions

- Each eigenvector has a direction - i.e. it is a dimension.
- Eigenvectors of symmetric matrices are orthogonal to each other - i.e. they point in completely different directions.
- Each eigenvalue is a number indicating how much variance there is in the data in that direction.
- Principal Components (PCs): New dimensions constructed as linear combinations of the original features, which are uncorrelated with one another. Constructed from eigenvectors.
- The first PC accounts for the most variability in the data. The next PC has the highest variance possible under the constraint that it is uncorrelated with the first PC, and so on...

Next lecture

Neural Networks

10th October 2023