Dimensionality Reduction (PCA)

Principal Component Analysis

Lesson 6: Scikit learn

An AI Commons initiative

Why do we need dimensionality reduction?

1. Can reduce the number of features to consider
   - saving memory and computational cost
   - generating better predictions
     - reduces overfitting
     - improves generalization (better results on new data)
2. Also helps simplify visualization
   - helps focus on the important features
   - helps in exploratory data analysis (EDA)

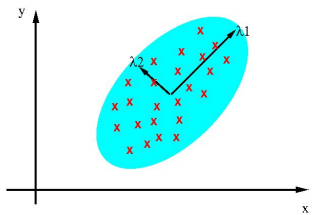
The curse of dimensionality

- Datasets often have a lot of features (dimensions)
- Data science and machine learning algorithms are statistical
  - they analyze distributions in a given space
- The number of features dictate the initial dimensions
  - imagine a dataset of n samples
  - the more dimensions (features) there is, the more apart these data points are from each other
  - the more apart they are, the harder it is to correlate them
- High dimensionality negatively impact algorithms
  - e.g. distance-based algorithms each like KNN become exhaustive
  - Ultimately, if every feature ends with one datapoint, the algorithm will overfit (learn just that datapoint and not generalize to others).

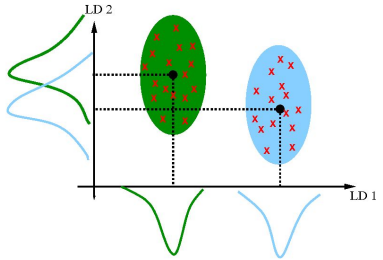
Dimensionality reduction algorithms

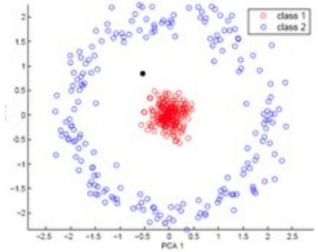
Algorithm	Reduction Concept	Applications
Principal Component Analysis	Find the axes with the greatest variance and project onto it Eliminate redundancy in correlated features	Unsupervised learning
Linear Discriminant Analysis	Find the axis with the maximum class separation and project onto it Use the labels to separate the distributions	Supervised learning
Kernel Principal Component Analysis	Modify the features space to provide a better separation	Unsupervised learning Denoising before further processing

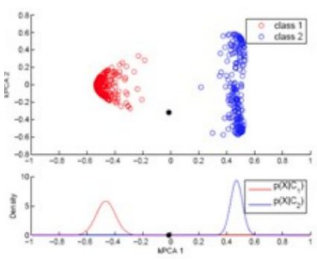
PCA: component axes that maximize the variance



LDA: maximizing the component axes for class−separation







Images Source

What is PCA?

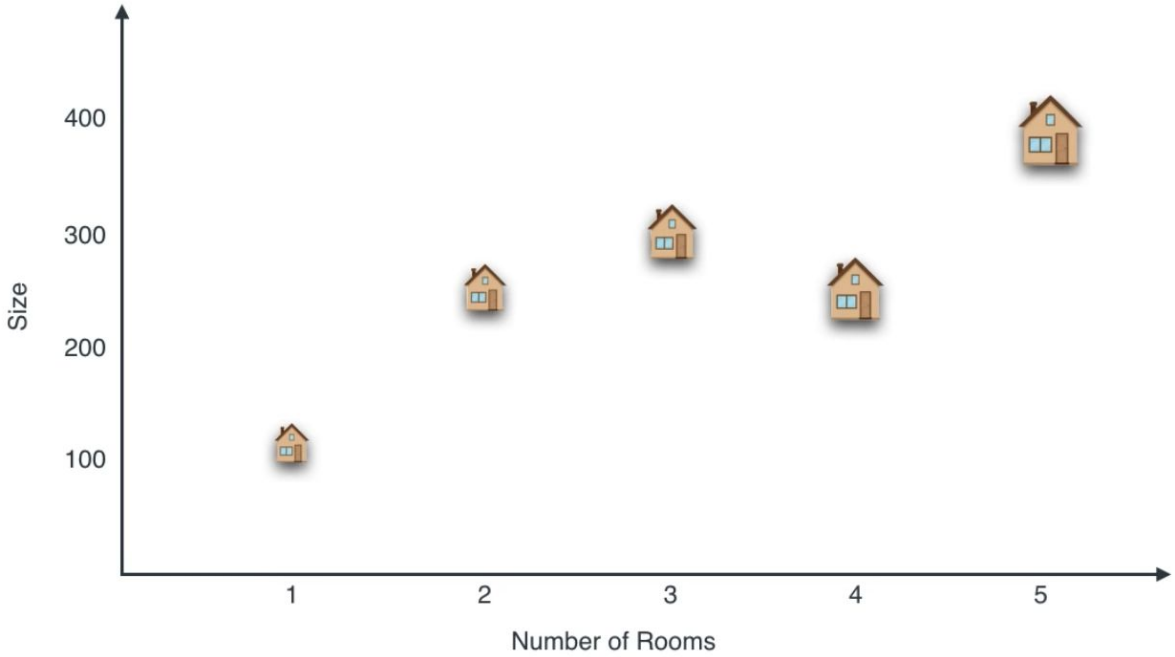
- Principal Component Analysis, or PCA, is an unsupervised learning **dimensionality-reduction** method
- PCA is used to reduce the number of features in a large dataset to a smaller set of variables while preserving as much information as possible
- PCA is unsupervised since we do not need to know the labels for each observation to reduce the number of features

Housing prices example

Toy Housing Data:

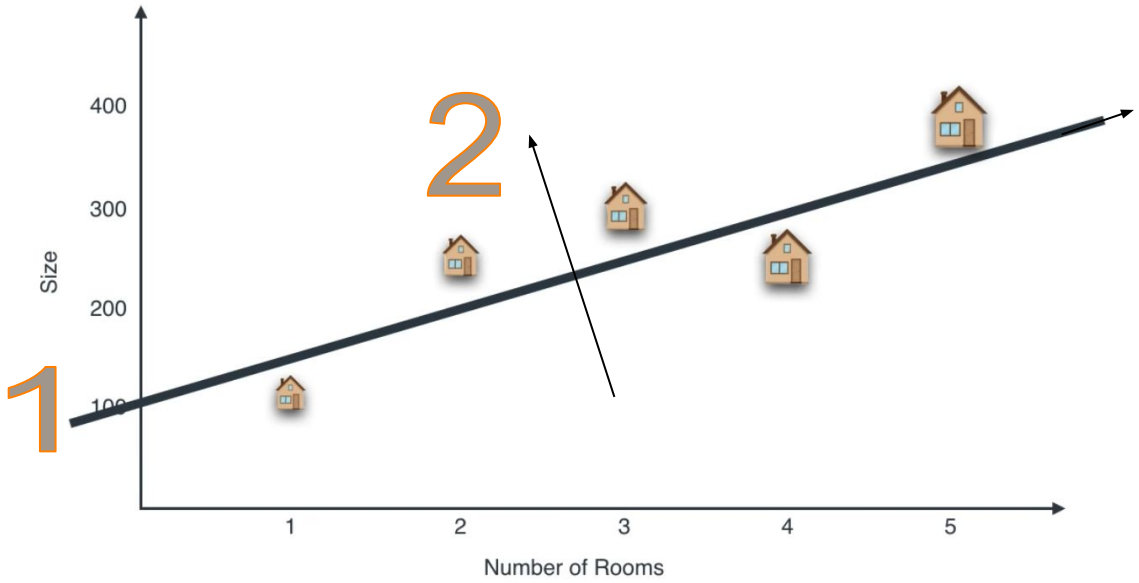
	Size	Number of Rooms
1	100	1
2	250	2
3	300	3
4	250	4
5	400	5

Visualizing the dataset features

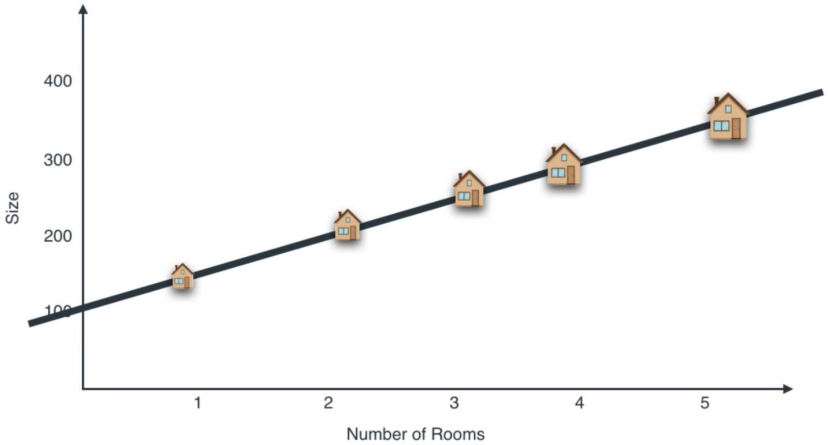


We have 5 observations (5 Houses), and each is defined by 2 features. Size and Number of Rooms

Variables can show interdependency



Reducing dimensions



⇒

Notice, how we now have one feature instead of two

PCA implementation in a nutshell

Learn ai

Data with many features ⇒ Standardize the data ⇒ PCA

⇓

We get top K **Principal Components** that retain the most information (maximize variance)

⇐ Use top K **Principal Components** to transform the data

⇐ Finally, we have new data with K features instead of the many features

AI

How does PCA work?

- Standardize the data
- Computing Covariance matrix
- Compute the Eigenvectors and Eigenvalues of the Covariance matrix and identify Principal Components
- Project the data based on the principal components

Step1: Standardization

- The aim is to standardize the range of the continuous initial variables so that each of them contributes equally to the analysis.

- PCA is sensitive to the variances of the initial variables.
  - Variables with larger ranges will dominate over those with small ranges
  - For example, a variable that ranges between 0 and 100 will dominate over a variable that ranges between 0 and 1
  - This would lead to biased results and transforming the data to comparable scales prevents this problem.

Step1: Standardization

Mathematically, subtracting the mean and dividing by the standard deviation for each value for each variable does the trick

$$normalized\ value = \frac{value - feature\ mean}{feature\ standard\ deviation}$$

Once the standardization is done, all the variables will be transformed to the same scale. This transforms the data into a common representation

StandardScalar Sklearn Library

```
>>> from sklearn.preprocessing import StandardScaler
>>> data = [[0, 0], [0, 0], [1, 1], [1, 1]]
>>> scaler = StandardScaler()
>>> print(scaler.fit(data))
StandardScaler()
>>> print(scaler.mean_)
[0.5 0.5]
>>> print(scaler.transform(data))
[[-1. -1.]
 [-1. -1.]
 [ 1.  1.]
 [ 1.  1.]]
>>> print(scaler.transform([[2, 2]]))
[[3. 3.]]
```

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

- **StandardScaler()** centres the data and usually does the job well but is prone to large outliers
- If there are large outliers, use **RobustScaler()** to remove the outliers.
- Then use **StandardScaler()** or **MinMaxScaler()**

Step 2: Covariance Matrix

Learn ai

$$\begin{bmatrix} Cov(x,x) & Cov(x,y) & Cov(x,z) \\ Cov(y,x) & Cov(y,y) & Cov(y,z) \\ Cov(z,x) & Cov(z,y) & Cov(z,z) \end{bmatrix}$$

Covariance Matrix for 3-Dimensional Data

The aim of this step is to understand how the variables of the input data set are varying from the mean with respect to each other, or in other words, to see if there is any relationship between them.

AI

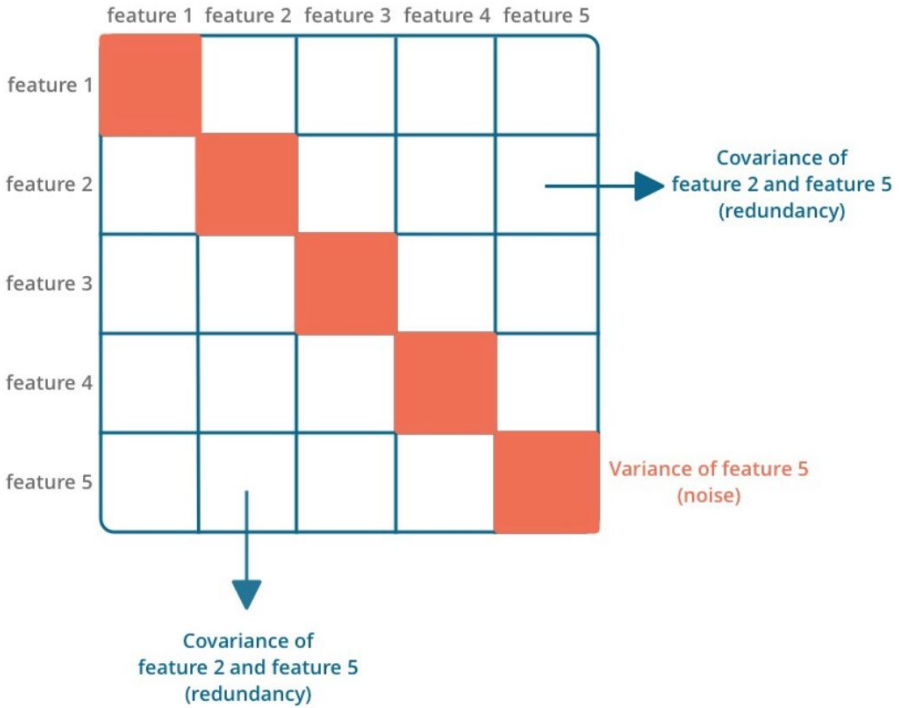
Step 2: Covariance Matrix

- It is a PxP matrix where P is the number of predictors/features/dimensions
- The elements along the diagonal are just the variances of each feature.
- This is the matrix that we extract the top K eigenvectors and eigenvalues out from which are known as our principal components.

$$\begin{bmatrix} Cov(x,x) & Cov(x,y) & Cov(x,z) \\ Cov(y,x) & Cov(y,y) & Cov(y,z) \\ Cov(z,x) & Cov(z,y) & Cov(z,z) \end{bmatrix}$$

Covariance Matrix for 3-Dimensional Data

Step 2: Covariance Matrix



Interpreting the covariance matrix.

Step 2: Covariance Matrix

Learn ai

- Understanding what role the covariance matrix plays in the PCA pipeline is important but plotting it is not of too much significance. “PCA” from Scikit Learn computes it so don’t stress too much since you don’t need to know how to compute it.
- Plotting it can help for a number of reasons (how? Next slide).
- You can use the code outlined here to do that

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt


# Load dataset
dataset = pd.read_csv('src/dataset.csv')


pca = PCA(dataset, standardize=True, method='eig')
normalized_dataset = pca.transformed_data


# Covariance Matrix
# bias =True, so dataset is normalized
# rowvar = False, each column represents a variable, i.e., a feature.
This way we compute the covariance of features as whole instead of
the covariance of each row


covariance_df = pd.DataFrame(data=np.cov(normalized_dataset,
bias=True, rowvar=False), columns=dataset.columns)


# Plot Covariance Matrix
plt.subplots(figsize=(20, 20))


sns.heatmap(covariance_df, cmap='Blues', linewidths=.7, annot=True,
fmt='.2f', yticklabels=dataset.columns)


plt.show()
```

AI

Step 2: Covariance Matrix

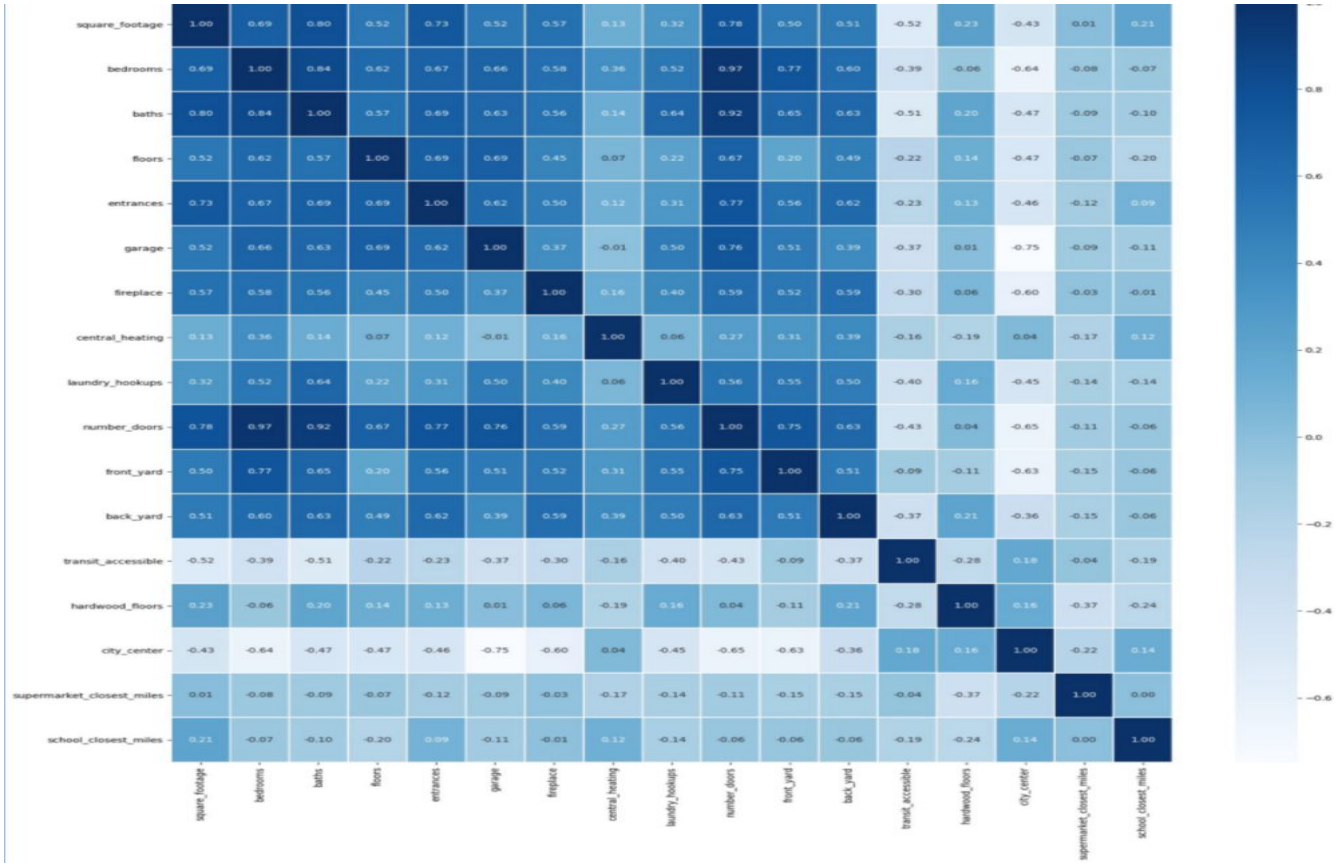
Learn ai

Why plotting the Covariance matrix helps:

- In the matrix, if Cov(feature1, feature2) = 0.95, we know these features have a strong linear relationship
- This means this pair of features encodes a similar pattern so we can consider this pair redundant (only one of them seems to be of value)
- We can notice the variance of each feature (diagonals), how high it is (more relevant it is) or how low it is (the less relevant it is)
- But this is just to judge the features, computing the covariance and jumping ship to step 3 is enough to go through the PCA pipeline by hand (Python implementation is simpler)

AI

Step 2: Covariance Matrix



Step 3: Compute Eigenvalues/Eigenvectors

Learn ai

- Take the covariance and compute all eigenvalues and eigenvectors
- With N initial features, there will N eigenvalue eigenvector pairs
- We arrange the eigenvalues in descending order and take the top K eigenvalues
- Since each eigenvalue is associated with an eigenvector, the corresponding eigenvectors become our top K principal components
- Remember K < N(total number of features).

AI

Step 4: Transform the data using the K principal components

- Now, we have K principal components
- We use these K Principal Components to transform/project the data onto a new space which is defined by K features
- As mentioned before, K < N (where N is the total number of features) and since our new data is defined by K features now as opposed to N features, we see the dimensions are reduced!
- This is new data with fewer features is the data we plug into a Machine Learning model

Python Implemented Example

Learn ai

Step1: Load the dataset & Split the dataset into Training and Test datasets

```
from sklearn.datasets import fetch_openml

mnist = fetch_openml('mnist_784')
```

```
from sklearn.model_selection import train_test_split

# test_size: what proportion of original data is used for test set
train_img, test_img, train_lbl, test_lbl = train_test_split(
mnist.data, mnist.target, test_size=1/7.0, random_state=0)
```

- This loads the MNIST dataset
- “Mnist.data” is a numpy array of shape (70000, 784)
- 70000 observations/data points & 784 features

AI

Step 2: Standardize the data:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# Fit on training set only.
scaler.fit(train_img)

# Apply transform to both the training set and the test set.
train_img = scaler.transform(train_img)
test_img = scaler.transform(test_img)
```

Step 3: Make an instance of the PCA model and fit the data

```
from sklearn.decomposition import PCA

# Make an instance of the Model
pca = PCA(.95)

pca.fit(train_img)
```

PCA(n_components=2) generates 2 principal components
PCA(0.95) retains 95% of the variance and generates however many principal components required to retain 95% variance

- After we “fit” train_img to the PCA model, ***pca.n_components_*** gives us the number of principal components

```
from sklearn.decomposition import PCA

pca = PCA(0.95)
pca.fit(train_img)
pca.n_components_
```

```
327
```

- 327 Principal components retain 95% of the variance

PCA(*n_components=None, *)*

n_components: how many principal components you want to use. This is where you specify the K value. If None is specified, all principal components are kept
OR
You specify the percentage of variance you want retained and the PCA automatically picks out the requisite number of principal components

pca.n_components_ : this gives us the principal components

Step 4: Finally, we use the “fitted” PCA model to map the data

```
train_img = pca.transform(train_img)
test_img = pca.transform(test_img)
```

- This gives us a new set of data with 327 features instead of the previous 384
- Our datasets are now ready to be used in a classification model to generate predictions of the MNIST data

https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html#sphx-glr-auto-examples-preprocessing-plot-scaling-importance-py

References

- https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
- https://scikit-learn.org/stable/datasets/toy_dataset.html
- https://scikit-learn.org/stable/auto_examples/decomposition/plot_pca_iris.html

Contributors

Mustafa Imam

Addison Weatherhead

Isha Sharma

Kevin Zhu

Gilles Fayad