Aleksandar Samsiev
Ivan Manov

Linear Algebra and Feature Selection

Course Notes

365√DataScience

Table of Contents

Abstract 5

Section 1: Linear Algebra Essentials 7

1.1 Why Linear Algebra? 7

1.2 Solving Quadratic Equations 8

1.3 Vectors 9

1.4 Matrices 10

1.5 The Transpose of Vectors and Matrices, the Identity Matrix 11

1.6 Linear Independence and Linear Span of Vectors 12

1.7 Basis of a Vector space, Determinant of a Matrix and Inverse of a Matrix 14

1.8 Solving Equations of the Form *Ax* = *b* 15

1.9 The Gauss Method 17

1.10 Other Types of Solutions of the Equation *Ax* = *b* 18

1.11 Determining Linear Independence of a Random Set of Vectors 18

1.12 Eigenvalues and Eigenvectors 19

1.13 Calculating Eigenvalues 20

1.14 Calculating Eigenvectors 21

Section 2: Dimensionality Reduction Motivation 23

2.1 Feature Selection, Feature Extraction, and Dimensionality Reduction23

2.2 The Curse of Dimensionality 24

Section 3: Principal Component Analysis (PCA) 27

3.1 An Overview of PCA 27

3.2 Step-by-step Explanation of PCA Through California Estates Example30

3.3 The Theory Behind PCA..32

3.4 PCA Covariance Matrix in Jupyter – Analysis and Interpretation.........33

Section 4: Linear Discriminant Analysis (LDA) 34

4.1 Overall Mean and Class Means 34

4.2 An Overview of LDA 35

4.3 LDA – Calculating Between- And Within-class Scatter Matrices 36

4.4 Step-by-step Explanation of Performing LDA on the Wine-quality Dataset 39

4.5 Calculating the Within and Between-Class Scatter Matrices................41

4.6 Calculating Eigenvectors and Eigenvalues for the LDA 43

4.7 Analysis of LDA...43

4.8 LDA vs PCA 44

4.9 Setting up the Classifier to Compare LDA and PCA 45

4.10 Coding the Classifier for LDA and PCA 46

4.11 Analysis of the Training and Testing Times for the Classifier and its Accuracy..47

Abstract

Linear algebra is often overlooked in data science courses, despite being of paramount importance. Most instructors tend to focus on the practical application of specific frameworks rather than starting with the fundamentals, which leaves you with knowledge gaps and a lack of full understanding. This course gives you an opportunity to build a strong foundation that would allow you to grasp complex ML and AI topics.

The course starts by introducing basic algebra notions such as vectors, matrices, identity matrices, the linear span of vectors, and more. These are used in solving practical linear equations, determining linear independence of a random set of vectors, and calculating eigenvectors and eigenvalues, all preparing you for the machine learning part of the matter - dimensionality reduction.

The concept of dimensionality reduction is crucial in data science, statistical analysis, and machine learning. This isn't surprising, as the ability to determine the important features in a dataset is essential - especially in today's data-driven age when one must be able to work with very large datasets.

Having hundreds or even thousands of attributes in your data could lead to a variety of problems – slow training time, the possibility of multicollinearity, the curse of dimensionality, or even overfitting the training data. Dimensionality reduction can help you avoid all these issues, by selecting the parts of the data which carry important information and disregarding the less impactful ones.

This course observes two staple techniques for dimensionality reduction – Principal Components Analysis (PCA), and Linear Discriminant Analysis (LDA). These methods transform the data you work with and create new features that carry most of the variance related to a given dataset.

Keywords: linear algebra, dimensionality reduction, feature selection, PCA, LDA

Section 1: Linear Algebra Essentials

Linear algebra describes the concepts behind the machine learning algorithms for dimensionality reduction. It builds upon vectors and matrices, linear equations, eigenvalues and eigenvectors, and more.

1.1 Why Linear Algebra?

Knowing linear algebra allows you to become a professional who understands the math on which algorithms are built, rather than someone who applies them blindly without knowing what happens behind the scenes.

Some of the most important skills required for achieving this goal:

- basic and advanced linear algebra notions
- solving linear equations
- determining independency of a set of vectors
- calculating eigenvalues and eigenvectors
- covariance matrix
- applying Principal Components Analysis
- applying Linear Discriminant Analysis
- performing dimensionality reduction in Python
- comparing the performance of PCA and LDA for classification with SVMs

1.2 Solving Quadratic Equations

By definition, a quadratic equation is an equation of second order with one unknown variable. “Order”, in this case, refers to the highest power of the unknown variable in the equation. In our case, that’s two.

A quadratic equation in its general form:

$$ax^2 + \mathrm{b}x + \mathrm{c} = 0$$

The letters a, b and c are constant coefficients, while x is the unknown variable.

Case scenarios regarding the number of possible solutions:

- the equation can have two distinct solutions, also called “roots of the equation”
- the equation can have a single solution – a double root
- the equation has no existing solutions at all

Based on a number called ‘discriminant’ we decide on the number of solutions the equation has. The formula for the discriminant of this equation looks like this:

$$D = b^2 - 4\mathrm{ac}$$

The discriminant is just a number computed by subtracting four times **a** times **c** from the square of the coefficient **b**. Based on it, we decide on *the number of solutions* the equation has.

- if the discriminant is a positive number, we have **two distinct solutions**

$$x_{1,2} = \frac{-b \pm \sqrt{D}}{2a}$$

- if D equals zero, the quadratic equation has a **single solution**

$$x = \frac{-b}{2a}$$

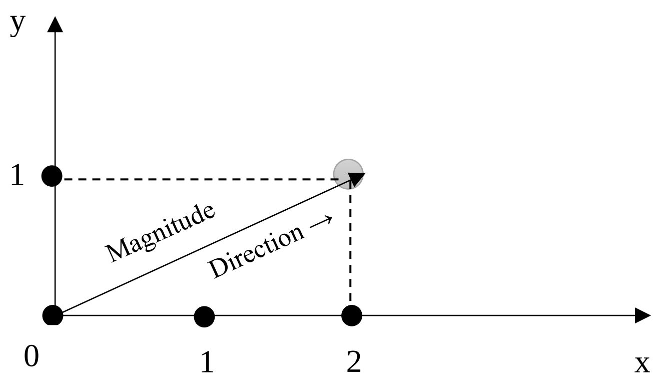
- if the discriminant is a negative number, then there are **no solutions** to the quadratic equation

1.3 Vectors

A vector is a one-dimensional object, characterized by *magnitude* and *direction*, containing numbers as elements. Geometrically, vectors are denoted by arrows whose length is the magnitude of the vector, and its direction is whichever way the vector is pointing to.

Vector types:

- row vectors [2 1]
- column vectors $\begin{bmatrix} 2 \\ 1 \end{bmatrix}$



Geometrical representation of a vector

Algebraic operations we can perform with vectors:

- addition
- subtraction
- calculating the dot product

Dot product – a scalar number obtained by executing a specific operation on the vector components:

$$p = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} \qquad q = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

$$p.q = p_1 \times q_1 + p_2 \times q_2 + p_3 \times q_3$$

Calculating a dot product

1.4 Matrices

Matrices are 2-dimensional objects, representing a collection of numbers in rows and columns. Matrices can be considered as a collection of vectors ordered as rows or columns.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

An example of a matrix

Algebraic operations we can perform with matrices:

- addition (the shape of matrix A must be the same as the shape of matrix B)

- subtraction (the shape of matrix A must be the same as the shape of matrix B)
- multiplication (the number of columns in matrix A must match the number of rows in matrix B)

Matrix multiplication - taking the dot product between each row vector in A and each column vector in B:

$$A \times \mathrm{B} = \begin{bmatrix} a_{11} \times b_{11} & + & a_{12} \times b_{21} & + & a_{11} \times b_{11} & + & a_{12} \times b_{22} \\ a_{21} \times b_{11} & + & a_{22} \times b_{21} & + & a_{21} \times b_{12} & + & a_{22} \times b_{22} \end{bmatrix}$$

Matrix multiplication

1.5 The Transpose of Vectors and Matrices, the Identity Matrix

Transposition - takes a matrix or a vector and transforms it into another matrix or vector.

Transpose of vectors - transforming row vectors into column ones and vice versa.

Transpose of matrices – transforming the row vectors in the matrix into columns.

$$A = \begin{bmatrix} 1 & 2 & 4 \\ 5 & 12 & 6 \end{bmatrix} \qquad A^T = \begin{bmatrix} 1 & 5 \\ 2 & 12 \\ 4 & 6 \end{bmatrix}$$

Transposing of a matrix

A^T – the transpose matrix

Note: a non-square matrix changes its shape when we apply the transpose function on it:

Multiplying a scalar by a vector or a matrix - we scale vectors and matrices by scaling each of their elements by this scalar

Identity matrix – a matrix that has ones on the diagonal and zeros elsewhere

$$I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

2 x 2 Identity matrix

If we have a m-by-n matrix A and multiply it by the n-dimensional identity matrix I_n, then we simply obtain A once again. Similarly, multiplying the m dimensional identity matrix with A gives us A again. That's why it is called the "identity matrix" – it acts as one in the integers.

$$\mathrm{A}_{\mathrm{mxn}} \times I_n = \mathrm{A}_{\mathrm{mxn}}$$

$$I_m \times \mathrm{A}_{\mathrm{mxn}} = \mathrm{A}_{\mathrm{mxn}}$$

1.6 Linear Independence and Linear Span of Vectors

Linear combination of set of vectors:

$$\lambda_1 v_1 + \lambda_2 v_2 + \cdots + \lambda_n v_n$$

v – vectors

λ – real numbers

Linear span - the set of all possible linear combinations of these vectors

Standard basis vector

Each of the standard basis vectors has a single value one and zeros elsewhere. The index of *e* indicates the position on which value *one* is located. If we're in a two-dimensional space, then e_1 is the vector $\begin{bmatrix}1\\0\end{bmatrix}$, and e_2 is $\begin{bmatrix}0\\1\end{bmatrix}$.

$$\alpha = \begin{bmatrix}-1\\2\end{bmatrix} \qquad e_1 = \begin{bmatrix}1\\0\end{bmatrix} \;\; e_2 = \begin{bmatrix}0\\1\end{bmatrix}$$

$$\alpha = -1 \times e_1 + 2 \times e_2$$

$$\alpha - a\ linear\ combination\ of\ e_1 and\ e_2$$

Linear independence

A set of vectors consists of linearly independent vectors when none of them are in the linear span of the rest vectors in this set. "*Independent*" means that not one vector in the set is a multiple of another. *"Linearly"* is derived from the fact that we perform linear combinations with the vectors in the rest of the set.

$$\mu_1 \times v_1 + \mu_2 \times v_2 + \cdots + \mu_n \times v_n = 0$$

Linear independent set of vectors

Linear dependence

A set of vectors is linearly dependent if there is a linear combination of them with non-zero coefficients that equals zero.

$$\mu_1 \times v_1 + \mu_2 \times v_2 + \cdots + \mu_n \times v_n = 0$$

Linear dependent set of vectors

1.7 Basis of a Vector space, Determinant of a Matrix and Inverse of a Matrix

Vector space –a set of vectors that can be added and subtracted together, as well as multiplied by numbers, called scalars.

Basis of a vector space - a set of vectors whose number equals the dimension of that space, or in other words - a set of vectors that are linearly independent of each other, and their linear span is the entirety of the vector space.

A *basis* means a set of vectors that form a vector space. Any vector in the vector space is in the span of the basis, which means that in order to obtain any vector from a certain vector space, it is sufficient to know its *basis vectors*, from which we make a *linear combination* of them to achieve that. You can think of a basis of a vector space as the *smallest set of vectors* that generates it.

A determinant of a matrix - a number that you can obtain from any square matrix. It characterizes some matrix properties – for example, whether it is invertible or not.

Note: not all matrices can be inverted.

If a matrix can be inverted, then we express this with the following mathematical equation:

$$\mathrm{A} \times \mathrm{A}^{-1} = I \text{ and } \mathrm{A}^{-1} \times \mathrm{A} = I \text{ where } \boldsymbol{I} \text{ is the } \boldsymbol{identity\ matrix}$$

A matrix is *invertible* if it's classified as a square matrix and its determinant does not equal zero. On the other hand, if a matrix is non-square, meaning the number of its rows does not match the number of its columns or its determinant is zero, then it is *non-invertible*.

Finding the inverse of a matrix:

1. Calculating its determinant by multiplying the entries on both diagonals and, afterward, subtracting the results from each other

$$H = \begin{bmatrix} 2 & 3 \\ 2 & 2 \end{bmatrix}$$

$$det(H) = 2 \times 2 - 2 \times 3 = -2$$

Determinant of a matrix

2. Filling the entries in the adjugate matrix using the technique of crossing off rows and columns

3. Calculating the inverse matrix with the help of the adjugate matrix and the determinant

$$H^{-1} = \frac{1}{det(H)} \times \begin{bmatrix} 2 & -3 \\ -2 & 2 \end{bmatrix}$$

$$H^{-1} = \begin{bmatrix} -1 & \frac{3}{2} \\ 1 & -1 \end{bmatrix}$$

Calculating the inverse matrix

1.8 Solving Equations of the Form *Ax* = *b*

Linear equations

$$ax + b = c$$

a, b, c – constant coefficients

x – variable to be determined

$$x = \frac{c - b}{a}$$

System of equations

$$a_1x_1 + a_2x_2 + a_3x_3 = y_1$$

$$b_1x_1 + b_2x_2 + b_3x_3 = y_2$$

$$c_1x_1 + c_2x_2 + c_3x_3 = y_3$$

This system can be represented as multiplication of matrices:

$$Ax = \mathrm{y}$$

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \mathrm{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

Note: if a matrix A is invertible, we can multiply both sides of our equation by A inverse from the left: $A^{-1} \times Ax = A^{-1} \times \mathrm{y}$ ***but we <u>cannot</u> multiply by A inverse from the right:*** $Ax \times A^{-1} = \mathrm{y} \times A^{-1}$.

$A^{-1} \times Ax = A^{-1}\mathrm{y}$, but $A^{-1} \times A = I$ – the identity matrix

$$\Rightarrow Ix = x$$

$$\Rightarrow x = A^{-1}\mathrm{y}$$

If the matrix A is non-invertible, we use a different approach – the Gauss method.

1.9 The Gauss Method

The Gauss method is a tool for solving linear equations. Also known as "Gaussian elimination", the method involves algebraic operations, aiming to eliminate as many as possible of the unknown variables in each row of the matrix, representing a system of linear equations.

Augmented matrix – a combination of the matrix A and the vector y from the equation $Ax = \mathrm{y}$.

$$\left[\begin{array}{ccc|c} a_{11} & a_{12} & a_{13} & y_1 \\ a_{21} & a_{21} & a_{23} & y_2 \\ a_{31} & a_{32} & a_{33} & y_3 \end{array}\right]$$

Augmented matrix

The goal is to find the vector *x* of unknown variables x_1, x_2, and x_3. So, the best scenario after the Gauss method is applied, is to obtain one of the three unknowns directly from one of the rows in the system while eliminating the rest two of the unknowns. Then, after we have one out of three variables found, we must substitute it in one of the other two rows in the system so that we can find one of the two unknown variables. In the end, we will use our third row to find the third and final unknown variable, using the two already known ones.

Gaussian elimination example:

$$\left[\begin{array}{cccc|c} 1 & -2 & 5 & 25 & 8 \\ -1 & 2 & -5 & -25 & -8 \\ 2 & -4 & 10 & 50 & 16 \\ 3 & -6 & 15 & 75 & 24 \end{array}\right] \frac{r_2 \to r_2 + r_1}{r_3 \to r_3 - 2 \times r_1} \to \left[\begin{array}{cccc|c} 1 & -2 & 5 & 25 & 8 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 3 & -6 & 15 & 75 & 24 \end{array}\right]$$

$$\left[\begin{array}{cccc|c} 1 & -2 & 5 & 25 & 8 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 3 & -6 & 15 & 75 & 24 \end{array}\right] r_4 \to r_4 - 3 \times r_1 \to \left[\begin{array}{cccc|c} 1 & -2 & 5 & 25 & 8 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

Hence solution vector *x* becomes:

$$\mathrm{x}_1 - 2 \times \mathrm{x}_2 + 5 \times \mathrm{x}_3 + 25 \times \mathrm{x}_4 = 8$$

1.10 Other Types of Solutions of the Equation *Ax* = *b*

Cases we can deal with regarding the solution of the equation: Ax=b:

- the equation has a unique solution - a single vector that solves the equation
- the equation has a general solution - several (or infinitely many) vectors that solve the equation
- the equation has no solutions

All these cases are determined after performing the Gauss method.

1.11 Determining Linear Independence of a Random Set of Vectors

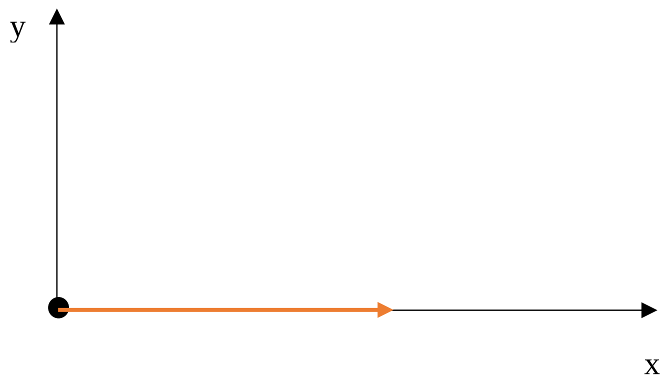
With the Gauss method's help, we can determine whether a given set of vectors is linearly independent or not. For this purpose, we must:

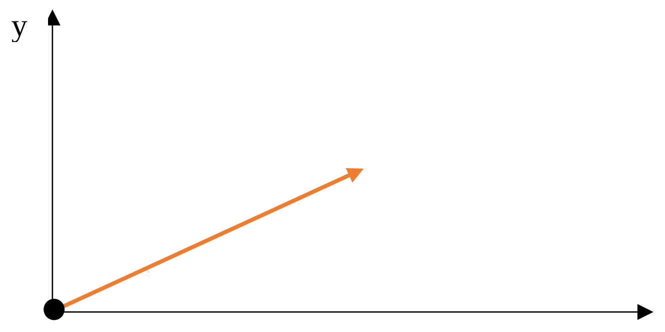
- build the augmented matrix

- perform Gaussian elimination
- transform the augmented matrix back to linear system form
- find the unknown variables in the equations and estimate the solution vector
- determine whether the set of vectors is linearly independent or not based on values in the solution vector

1.12 Eigenvalues and Eigenvectors

Eigenvectors of a matrix - non-zero vectors that can change by a scalar factor when we apply a certain *linear transformation* which, in turn, is the multiplication by its matrix.





Any transformation performed onto that vector is described via a matrix – the rotation matrix.

The rotation matrix

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & cos(\theta) \end{bmatrix}$$

Eigenvalue - the scalar factor by which the matrix scales the eigenvector.

$A = \begin{bmatrix} \mathrm{a} & \mathrm{b} \\ \mathrm{c} & \mathrm{d} \end{bmatrix}$

v – Eigenvector of the matrix A

λ – Eigenvalue

$\mathrm{Av} = \lambda\mathrm{v}$ – characteristic equation – an equation for obtaining the eigenvalues of a matrix

1.13 Calculating Eigenvalues

To obtain the eigenvalues of a matrix, we must solve an equation called the "characteristic equation": $\mathrm{Av} = \lambda\mathrm{v}$. Its solutions represent the eigenvalues of the matrix. The characteristic equation is obtained from the following equation:

$$det(\mathrm{A} - \lambda I) = 0$$

A – Matrix A

I – Identity matrix

λ – Eigenvalue

The solutions of the characteristic equation represent the *eigenvalues* of the matrix A.

1.14 Calculating Eigenvectors

Each eigenvalue can correspond to a single eigenvector, or to many eigenvectors. If a matrix A has an eigenvector *v* with an associated eigenvalue λ, the following equation holds:

$$\mathrm{Av} = \lambda \mathrm{v}$$

which can also be rewritten this way:

$$(\mathrm{A} - \lambda I)\mathrm{v} = \mathbf{0}$$

To calculate the eigenvalues, we must:

- determine $\mathrm{A} - \lambda I$ by replacing λ with the value of an already known eigenvector
- obtain an eigenvector v from the resulting equation
- apply the Gauss method using the same approach as with the equation ***Ax = b***
- transform the obtained matrix into a linear system
- find the eigenvectors corresponding to the respective eigenvalues

Section 2: Dimensionality Reduction Motivation

2.1 Feature Selection, Feature Extraction, and Dimensionality Reduction

Definitions:

Feature selection - the process of reducing the number of features used in various types of machine learning models in a way that leaves the predictive ability of the data preserved

Features - the transformed raw data after a process called "feature engineering"

Feature engineering - the stage which prepares the input dataset and makes it compatible with the machine learning algorithms

Features

	Square footage	**Room count**	**Window count**
House 1	87	3	6
House 2	94	4	7

Observations (samples)

A dataset example containing features and observations

High-dimensional data – a dataset where the number of features exceed the number of observations drastically

High-dimensional data often lead to various problems which can impact the performance of the machine learning algorithm, making it unreliable. A set with many features probably means that the volume is large, but the data points are very few and far apart and this is problematic for the algorithm. In such cases, we could try to remove the irrelevant features without losing essential information from the dataset. Reducing the features can boost the training accuracy of the model, hence its predictive power. By applying feature selection, we examine the key features that affect our model, removing the unnecessary ones as a result.

Data sparsity – cases when the volume of a high-dimensional dataset is very large, while the data points in it are few and far apart

Feature extraction – the process of transforming existing features into new ones – linear combinations of the originals

Dimensionality reduction = feature selection + feature extraction

When applying *feature selection*, working with a subset of the original data indicates that we'll have fewer features compared to the original dataset, hence the dimensional space would be lower. In *feature extraction*, although the number of the newly constructed features might be the same as the original, we usually use only the most significant ones. In particular, evaluating the retained variance in the dataset helps us select those features that preserve the most information for our data.

2.2 The Curse of Dimensionality

The phenomenon describes a set of problems that arise when working with a high-dimensional dataset. The phrase "curse of dimensionality" was coined by the mathematician Richard Bellman and refers to the difficulties we face when optimizing a function with many input variables. That said, the error can increase when adding dimensions, or similarly, features to the dataset.

Richard Bellman

Troubles related to the curse of dimensionality:

- Lack of data points in comparison to the high dimensionality of the model
- Huge distance between the data points
- Increased complexity caused by the growing number of possible combinations of features
- Possibility of including more noise in the training data
- Problems with the data storage
- Training taking more time – larger input datasets increase the computational complexity leading to longer training periods

Dealing with the curse of dimensionality = applying dimensionality reduction

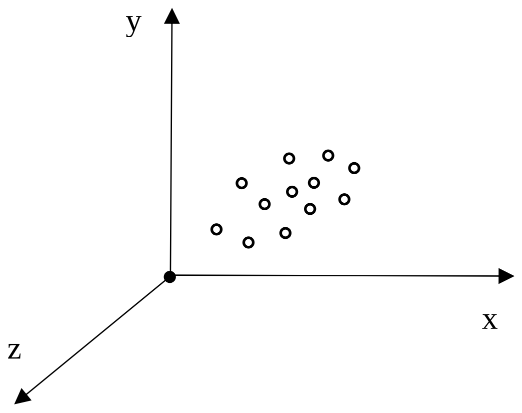
To battle the issue of large and sparse datasets, we use dimensionality reduction to close up the feature space. This method contains no additional inputs, which makes the data analysis a straightforward process for the machine learning algorithm.

Section 3: Principal Component Analysis (PCA)

3.1 An Overview of PCA

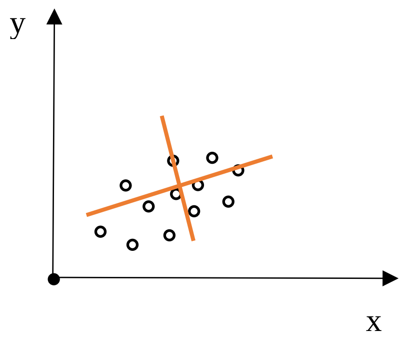
Principal Component Analysis, also known as PCA is a feature extraction algorithm used for dimensionality reduction. It is one of the oldest and most widely used dimensionality reduction techniques in unsupervised learning. PCA is used to reduce the number of features in a dataset into a smaller subset which preserves most of the information from the original set.

- Works mainly with numerical data
- Doesn't require too much information about the data



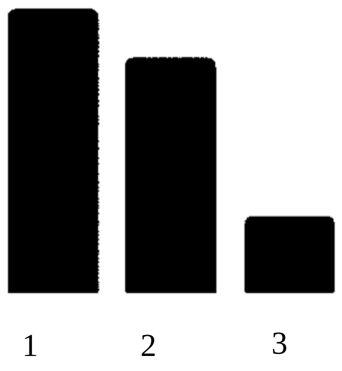
Three-dimensional dataset

The PCA algorithm constructs new axes, called principal components (PCs), whose number equals the number of variables in our initial dataset. They are not the same as the original features - these principal components capture the most variance of the data in each direction. The goal is to capture the most valuable information in the data.



Two-dimensional dataset

These axes are ordered by the amount of retained variance in the data. Therefore, the first few components carry the most information about our data. That means we could discard several less important components, without experiencing a significant loss of information.



Principal Components

Retaining at least 80% of the original variance, usually means we've kept the majority of important information. When we analyze the principal components, we are normally interested in calculating and plotting metrics, such as variance captured by each separate component, as well as cumulative variance. Plotting these can help us decide how many of the principal components we are going to use and which ones to discard. After we make this decision and we project our standardized data onto the new axes (the PCs), we lower the dimension of our data, so dimensionality reduction occurs.

PCA constructs the principal components so that they are uncorrelated, meaning there is no linear relationship between them. Therefore, the components are constructed in a way that makes the next component perpendicular to the previous ones. We want axes that separate the points clearly, possibly distinguishing the groups of points better than the original axes do. PCA will find these axes as the dimension of our data.

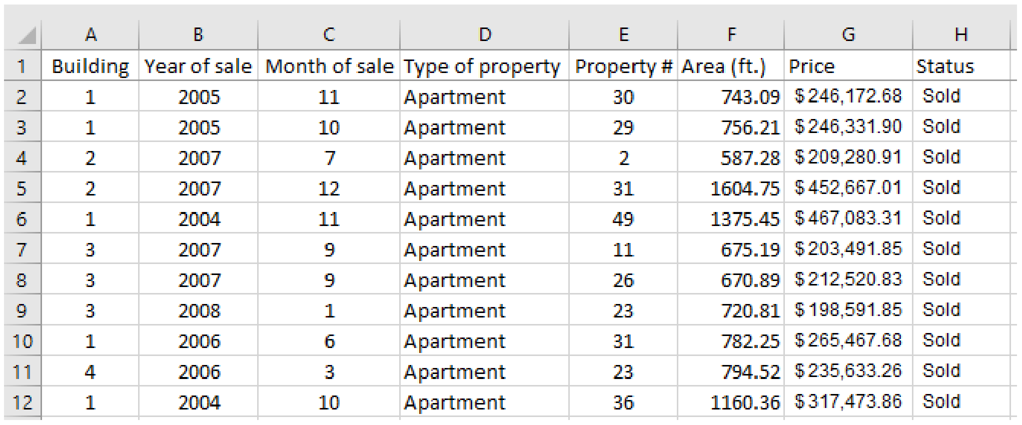
Note: when the data we work with has only two explanatory variables, we can decide to take both components produced by PCA. That won't lower the dimension, but rather give a more distinguishable view of the data.

When our data is higher dimensional, after all the PCs have been constructed by PCA, if we want to lower the dimension, we need to project our data points onto a subset of them. This subset will be carrying enough variance for the data. The dimension is lower because we have chosen only the first few PCs as mentioned before. This way our new axes are less than the original ones.

3.2 Step-by-step Explanation of PCA Through California Estates Example

The explanation consists of a practical example observed in Jupyter Notebook. Here, we concentrate on the main steps describing the process, rather than analysing the code. The code itself can be fully observed in the course content.

1. Exploring the data



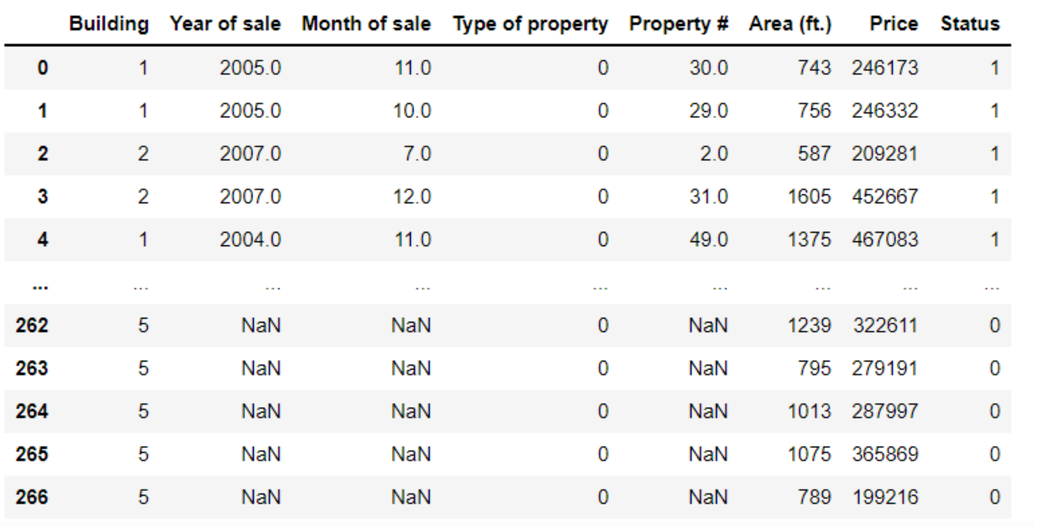
	A	B	C	D	E	F	G	H
1	Building	Year of sale	Month of sale	Type of property	Property #	Area (ft.)	Price	Status
2	1	2005	11	Apartment	30	743.09	$246,172.68	Sold
3	1	2005	10	Apartment	29	756.21	$246,331.90	Sold
4	2	2007	7	Apartment	2	587.28	$209,280.91	Sold
5	2	2007	12	Apartment	31	1604.75	$452,667.01	Sold
6	1	2004	11	Apartment	49	1375.45	$467,083.31	Sold
7	3	2007	9	Apartment	11	675.19	$203,491.85	Sold
8	3	2007	9	Apartment	26	670.89	$212,520.83	Sold
9	3	2008	1	Apartment	23	720.81	$198,591.85	Sold
10	1	2006	6	Apartment	31	782.25	$265,467.68	Sold
11	4	2006	3	Apartment	23	794.52	$235,633.26	Sold
12	1	2004	10	Apartment	36	1160.36	$317,473.86	Sold

California_Real_Estate_CSV.csv

The dataset contains 8 variables, 6 of which are numerical. PCA works with numeric data, so we need to transform the categorical variables into numerical by using dummy variables. The dataset needs to be preprocessed first.

2. Transformation of the data

The NaN values which appear after loading the data in Python must be discarded.



	Building	Year of sale	Month of sale	Type of property	Property #	Area (ft.)	Price	Status
0	1	2005.0	11.0	0	30.0	743	246173	1
1	1	2005.0	10.0	0	29.0	756	246332	1
2	2	2007.0	7.0	0	2.0	587	209281	1
3	2	2007.0	12.0	0	31.0	1605	452667	1
4	1	2004.0	11.0	0	49.0	1375	467083	1
...	...	...	...	...	...	...	...	...
262	5	NaN	NaN	0	NaN	1239	322611	0
263	5	NaN	NaN	0	NaN	795	279191	0
264	5	NaN	NaN	0	NaN	1013	287997	0
265	5	NaN	NaN	0	NaN	1075	365869	0
266	5	NaN	NaN	0	NaN	789	199216	0

- Standardizing the data – rescaling:

*We want the algorithm to interpret all variables **equally**.*

Mean = 0

Standard deviation = 1

- Constructing the principal components
- Analysing the variance captured by each principal component

Cumulative variance – the variance in the data captured up until a certain component

- Choosing a subset of principal components which captures enough variance – at least 80% of the total variance

3.3 The Theory Behind PCA

Covariance matrix – a matrix that shows the correlation between any pair of variables in our initial dataset

If our data has "n" dimensions the associated covariance matrix will have shape **$n_x n$**.

8 dimensions => 8_x8 covariance matrix

$$C=\begin{pmatrix} Cov(X^1,X^1) & Cov(X^1,X^2) & \cdots & Cov(X^1,X_n) \\ \vdots & \ddots & & \vdots \\ Cov(X_n,X^1) & \cdots & & Cov(X_n,X_n) \end{pmatrix}$$

*General form of the covariance matrix for a dataset with n variables **x***

The covariance between two variables is symmetric - the covariance between X_1 and X_2 is the same as the covariance between X_2 and X_1.

$$Cov(X^1,X^2) = Cov(X^2,X^1)$$

Each of the values of a $n_x n$ covariance matrix is calculated by the formula:

$$Cov(X_i,X_k) = \frac{1}{n-1}\sum_{m=0}^{n}(X_m - \bar{X})(Y_m - \bar{Y})$$

$\bar{X}$ and $\bar{Y}$ - the ***means*** of the two variables respectively

The eigenvectors of the covariance matrix are the principal components.

➔ To find them, we must solve the eigenvector/eigenvalue problem - *Eigendecomposition*

$$det(C - \mu I) = 0$$

C – covariance matrix

I – identity matrix

μ - the unknown variable

Having obtained the eigenvalues, we solve the linear system $\mathbf{Cx} = \mu\mathbf{x}$ for every eigenvalue $\boldsymbol{\mu}$ and we find the eigenvector $\boldsymbol{x}$ for the corresponding eigenvalue $\boldsymbol{\mu}$. After determining all eigenvector- eigenvalue pairs, we can calculate the variance carried in each principal component from the eigenvalues. By ranking our eigenvectors for their eigenvalues from highest to lowest, we obtain the principal components in order of significance. To compute the percentage of information accounted by each component, we divide the *eigenvalue of the respective component* by *the sum of all eigenvalues*.

3.4 PCA Covariance Matrix in Jupyter – Analysis and Interpretation

After determining the number of principal components, we will use in PCA, we must interpret these components and their relationship to the original features. Since each component is a combination of their initial variables, we must find those with the biggest weights in the equation of each component, or, in other words, which variables describe each of our components best.

- Loading the correlation between each component and each of our variables
- Analysing the correlations and explaining the principal components via the initial variables – interpreting the components
- Preparing the principal components to be the new axes
- Projecting the standardized data points onto the PCs

Section 4: Linear Discriminant Analysis (LDA)

4.1 Overall Mean and Class Means

Sample *Features* *Category*

Student	Math	English	Gender
Jack	50%	31%	Male
Jessica	73%	80%	Female
James	62%	75%	Male
Georgia	81%	90%	Female

A dataset example containing students tests results

The length of the overall and the class means corresponds to the number of features.

In this case the features are two,

➔ The overall and the class means vectors will be of length *two*.

Overall mean

$\begin{bmatrix} x \\ x \end{bmatrix}$ the first entry is the overall mean for the 1st feature – the Math exam scores

$\begin{bmatrix} x \\ x \end{bmatrix}$ the second entry is the overall mean of the 2nd feature – the English exam scores

Class means

Classes = number of class means

Mean (male) = $\begin{bmatrix}56\\53\end{bmatrix}$ $\frac{50+62}{2} = 56$ $\frac{31+75}{2} = 53$

Math score English score

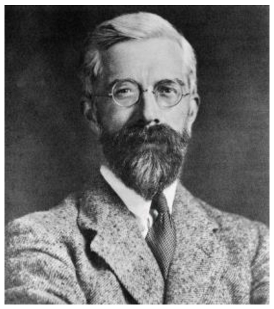
Mean (female) = $\begin{bmatrix}77\\85\end{bmatrix}$ $\frac{73+81}{2} = 77$ $\frac{80+90}{2} = 85$

Math score English score

4.2 An Overview of LDA

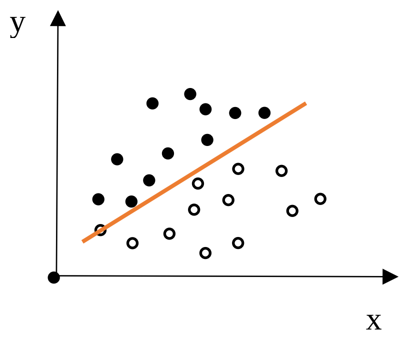
LDA is a supervised learning algorithm used for dimensionality reduction. The goal of LDA is to find a linear combination of features to best separate two or more classes. LDA aims to:

- maximize the distances between the classes
- minimize the variance (scatter) of data within each class



Ronald Fisher

When performed, LDA constructs new axes, called linear discriminants, where the data is being projected. In LDA, the number of linear discriminants is, at most, *c-1*, where *c* is the number of distinct classes in our data. We are interested in constructing axes that better separate the data points according to their labels and into well-formed clusters.



LDA Overview

4.3 LDA – Calculating Between- And Within-class Scatter Matrices

S_W – Within-class scatter matrix

S_B – Between-class scatter matrix

When applied, LDA finds the between- and within-class scatter matrices by using the class means and the overall data mean. Then, LDA finds the eigenvalues and the eigenvectors of the matrix obtained from multiplying the inverse of S_W with

S_B. These eigenvectors are the linear discriminants, or the axes of our low-dimensional space.

$$\frac{(\mu^1-\mu^2)^2}{s^{12}+s^{22}}$$ - Fisher's Discriminant Ratio

Fisher's Discriminant Ratio represents the idea behind LDA: *maximizing the distance between the class means, while minimizing the scatter.* Considering the ratio, this means *maximizing the numerator, while minimizing the denominator*.

Multi-class data – dataset containing more than two class labels

When we deal with multi-class data, we use the following approach:

- Calculating the overall mean of the data – centroid
- Measuring the distance from each class mean to the centroid
- Maximizing the distance between each class mean and the centroid while minimizing the scatter for each category:

$$\frac{d_1^2+d_2^2+d_3^2}{s_1^2+s_2^2+s_3^2}$$

d_1^2 – the distance between the centroid and the mean of the class 1

s_1^2 – the scatter for the class 1

Calculating the between-class scatter matrix:

$$S_B=\sum_{i=1}^{c} N_i(\mathrm{m_i}-\mathrm{m})(\mathrm{m_i}-\mathrm{m})^T$$

$\mathrm{N_i}$ – the number of points (observations) in the ith class

$\mathrm{m_i}$ – the mean vector of the ith class m – the mean vector of the whole dataset

i – the number of classes

c – the class

With the between-class scatter matrix we measure how far away the single class clusters are.

Calculating the within-class scatter matrix:

$$S_w = \sum_{i=1}^{c} S_i$$

S_i – the covariance matrix for the $\mathrm{i^{th}}$ class

$$S_i = \sum_{x \epsilon C_i} (\mathrm{x} - \mathrm{m_i})(\mathrm{x} - \mathrm{m_i})^T$$

The within-scatter matrix represents the sum of the covariance matrices for each separate class. Each class covariance matrix shows the relations between the features in this class.

Covariance matrix – a symmetric square matrix giving the covariance between each pair of elements

Rewriting Fisher's ratio:

$$\widetilde{w} = \underset{v}{argmax} \frac{v^T S_B v}{v^T S_w v}$$

maximizing the between-class scatter S_B, and minimizing the within-class scatter S_w

$\widetilde{w}$ - the direction which gives maximum class separability of the dataset

This ratio can also represent the eigenvalue problem given by the following formula:

$$S_w v = \mu S_B v$$

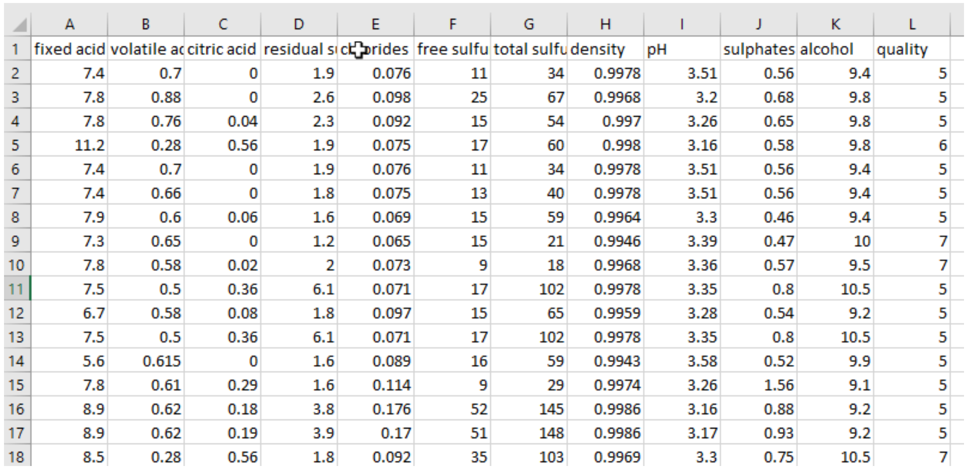
We need to find the eigenvalues and their corresponding eigenvectors of the matrix, produced by the multiplication between the inverse within-scatter matrix with the between-scatter matrix.

After finding the eigenpairs, we must arrange the eigenvalues from largest to smallest, and reorder their eigenvectors accordingly. The number of linear discriminants we will use when performing LDA depends on the selecting of the most important ones – those with the highest eigenvalues.

4.4 Step-by-step Explanation of Performing LDA on the Wine-quality Dataset

The explanation consists of a practical example observed in Jupyter Notebook. Here, we concentrate on the main steps describing the process, rather than analyzing the code. The code itself can be fully observed in the course content.

- Exploring the data



	A	B	C	D	E	F	G	H	I	J	K	L
1	fixed acid	volatile ac	citric acid	residual s	chlorides	free sulfu	total sulfu	density	pH	sulphates	alcohol	quality
2	7.4	0.7	0	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5
3	7.8	0.88	0	2.6	0.098	25	67	0.9968	3.2	0.68	9.8	5
4	7.8	0.76	0.04	2.3	0.092	15	54	0.997	3.26	0.65	9.8	5
5	11.2	0.28	0.56	1.9	0.075	17	60	0.998	3.16	0.58	9.8	6
6	7.4	0.7	0	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5
7	7.4	0.66	0	1.8	0.075	13	40	0.9978	3.51	0.56	9.4	5
8	7.9	0.6	0.06	1.6	0.069	15	59	0.9964	3.3	0.46	9.4	5
9	7.3	0.65	0	1.2	0.065	15	21	0.9946	3.39	0.47	10	7
10	7.8	0.58	0.02	2	0.073	9	18	0.9968	3.36	0.57	9.5	7
11	7.5	0.5	0.36	6.1	0.071	17	102	0.9978	3.35	0.8	10.5	5
12	6.7	0.58	0.08	1.8	0.097	15	65	0.9959	3.28	0.54	9.2	5
13	7.5	0.5	0.36	6.1	0.071	17	102	0.9978	3.35	0.8	10.5	5
14	5.6	0.615	0	1.6	0.089	16	59	0.9943	3.58	0.52	9.9	5
15	7.8	0.61	0.29	1.6	0.114	9	29	0.9974	3.26	1.56	9.1	5
16	8.9	0.62	0.18	3.8	0.176	52	145	0.9986	3.16	0.88	9.2	5
17	8.9	0.62	0.19	3.9	0.17	51	148	0.9986	3.17	0.93	9.2	5
18	8.5	0.28	0.56	1.8	0.092	35	103	0.9969	3.3	0.75	10.5	7

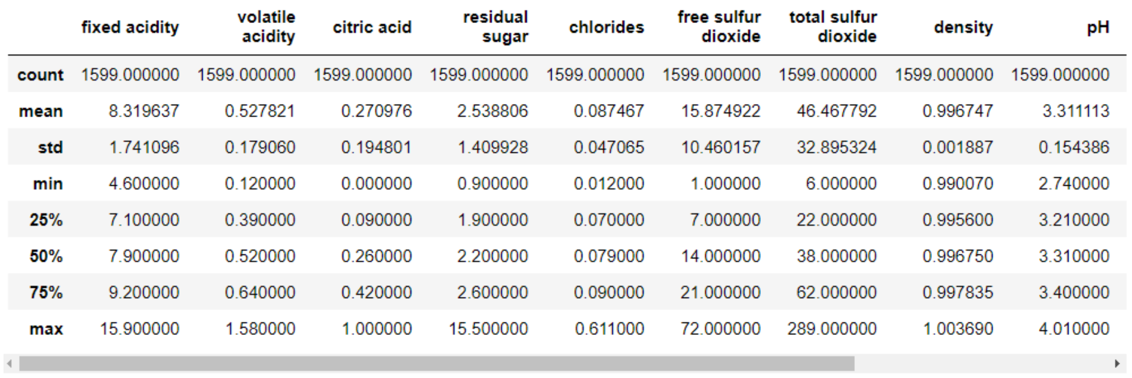
winequality.csv

Since our data is numerical, one thing we should analyze is the range of values each column takes. In particular, the *quality* column that shows how many different grades for wine quality we have, as well as what actual grades there are.

Our goal will be to predict the wine grades, which means that this will be our predictor column.

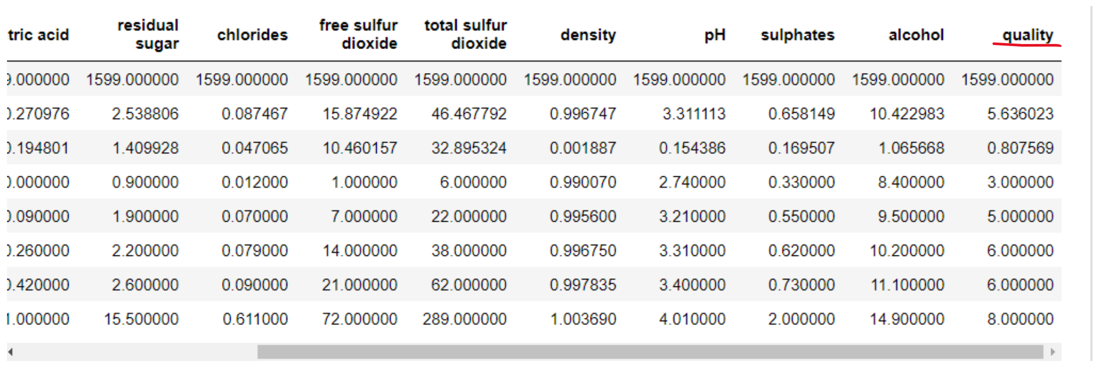
We see that the quality ranges from 3 to 8, therefore there are 6 distinct classes for our data.

- Transformation of the data



	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000

The data consists of 12 columns overall, 11 of which are feature columns. The last one, called *quality*, is the column that represents the data labels. Our goal is to separate the wine samples from the data and group them into clusters, according to their **quality grade**.



tric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113	0.658149	10.422983	5.636023
.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.169507	1.065668	0.807569
.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000	8.400000	3.000000
.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.550000	9.500000	5.000000
.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.620000	10.200000	6.000000
.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.730000	11.100000	6.000000
.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000	14.900000	8.000000

Each row in the data represents a particular wine, while each column describes certain characteristics. We have acidity levels, PH levels, as well as alcohol content, and plenty more.

- Standardizing the data – rescaling

The *quality* grades have no numerical influence on the data, instead they simply serve as *labels*, which is why we don't standardize them.

- Calculating the class mean – the class mean for each separate quality grade of wine
- Reading and interpreting the results

4.5 Calculating the Within and Between-Class Scatter Matrices

We calculate the within- and between-scatter matrices using Python in order to find the product' eigenvalues and eigenvectors of the inverse of the within-scatter matrix and the between-scatter matrix, so that we can make a feature selection and lower the dimension of our dataset.

- Calculating the covariance matrix for each separate quality grade and then summing those individual class covariance matrices up, resulting in a within-class scatter matrix based on the formula:

$$S_w = \sum_{i=1}^{c} S_i$$

Within-scatter class matrix

- Calculating the between-class scatter matrix
    1) Finding the overall mean
    2) Finding the class means
    3) Imposing the formula

$$S_B = \sum_{i=1}^{c} N_i(\mathrm{m_i} - \mathrm{m})(\mathrm{m_i} - \mathrm{m})^T$$

Between-scatter class matrix

- Multiplication between the inverse within-scatter matrix with the between-scatter matrix

$$S_w^{-1} S_B$$

4.6 Calculating Eigenvectors and Eigenvalues for the LDA

By finding the eigenvectors and ordering them by level of importance, we'll construct the linear discriminants - our new axes. These new axes will comprise our lower-dimensional space onto which we'll project the data points. In Python, this is a straightforward process.

- First, by using the method *.eig()* of the submodule *linalg* in NumPy and providing the matrix product as arguments, we can get the eigenvalues and eigenvectors
- Next, we need to pair and sort them in descending order, according to the eigenvalues - we divide each eigenvalue by the sum of all eigenvalues and multiply that by 100. This will give us the percentage of variance explained by the eigenvector associated with this eigenvalue

4.7 Analysis of LDA

To find the newly constructed axes' significance, we must add all eigenvalues and divide each of them separately by the overall sum to obtain the percentage of retained variance.

We must *project* the data onto the discriminants that make up more than 80% of the total variance. That's where dimensionality reduction occurs– instead of the original number of features, we'll have less (in this case – two) new linear discriminants that correspond to the two highest eigenvalues.

In the observed practical example, the first two linear discriminants make up around 95% of the total variance in the data. So, instead of the original eleven features, we'll have our two new linear discriminants that correspond to the two highest eigenvalues. We've now performed dimensionality reduction.

4.8 LDA vs PCA

- Comparing dimensionality reduction on the same dataset performed by both LDA and PCA
- Splitting the dataset into training and testing parts and applying apply LDA and PCA consecutively
- After the data is projected onto the linear discriminants in the case of LDA, and onto the principal components in the case of PCA - training and testing the classifier on these newly obtained datasets
- Timing the classifier for the training and testing part for both LDA and PCA
- Running the functions which train and test the classifier on the projected datasets from both dimensionality reduction techniques to compare the average training and testing times, assessing whether the classifier is faster when we perform LDA or PCA as a preliminary step
- Comparing the accuracy of the classifier by calculating a confusion matrix for the results in both approaches

- Analysing the results

Analysing the results confirm that there is a much better separation of the data with the LDA plot in terms of the ***quality grade*** *given.*

4.9 Setting up the Classifier to Compare LDA and PCA

Why do we split the initial features dataset into training and testing instead of making use of the already standardized dataset?

We must first start with discussing the standardization tool - "Standard Scaler" and the three applied methods:

- *.fit()* - takes the dataset we aim to standardize as an argument and computes its mean and standard deviation. These will then be used in the formula to rescale each feature. In the case of the Standard Scaler this transform each feature to have a mean of 0 and a standard deviation of 1.
- *.transform()* - Applies the rescaling formula to every feature and transforms them. In this case the resulting features have a zero mean and standard deviation 1.
- *.fit_transform()* - applies both methods – fit and transform

Answer: using *.fit transform()* on the train data results in standardizing it. The method calculates the mean and standard deviation for each feature. We use *.transform()* on the test data only. This means that we're applying a formula where **μ** is the mean of the training set for each feature, while **σ** is the standard deviation of the training set of each feature. Thus, we apply the metrics calculated from the training set onto the test set. The testing set aims to serve as unseen data, to model real-life scenarios.

4.10 Coding the Classifier for LDA and PCA

- creating an instance of the class LDA with two linear discriminants
- fitting the training data
- fitting the testing data
- creating an instance of the class PCA with two principal components
- fitting the training data
- fitting the testing data
- importing the support vector classifier
- training and testing the classifier, while timing the code
- predicting the time of using the classifier for both algorithms
- comparing the accuracy of the classifying model on both datasets

4.11 Analysis of the Training and Testing Times for the Classifier and its Accuracy

Ideally, we'd like to time our code for both LDA and PCA and figure out how accurate each analysis is. To make it more precise, we can run the functions for training and testing PCA and LDA a grand total number of ten times and take the average for training and testing in both functions.

Analysing the results:

The results for training times, as well as the predicting ones, are very close. However, the real metric of importance is the classifier's accuracy. But before that, we need to emphasize another metric – the confusion matrix.

Confusion matrix - a square matrix that shows which samples were graded correctly and which were not. In our case, it shows us the number of wines that we correctly identified in class 3, as well as the number of wines, which are in class 3, but we've misclassified. The same goes for the wines in class 4 through 8, as well. We use the confusion matrix because it helps us improve our model by analysing the misclassified data and adjusting the parameters of the model accordingly. It shows a deeper level of measuring performance compared to a simple accuracy measurement.

After the confusion matrix is ready, we'll measure the accuracy by calling "*accuracy_score*" on the correct and predicted quality grade values

Results:

Accuracy for LDA: 63%

Accuracy for PCA: 54%

LDA outperformed PCA with 9%. As expected, LDA also did much better when preparing labelled data for classification.

Aleksandar Samsiev
Ivan Manov

Email: team@365datascience.com

365√DataScience