Engineering and Applied Science Programs for Professionals
Whiting School of Engineering
Johns Hopkins University
685.621 Algorithms for Data Science

Data Processing

This document provides a rollup of the data processing methods covered in this module. The following list of topics is covered in this module allowing for an understanding of how data is processed to be used in a compact efficient manner for Data Science.

Contents

1 Introduction to Data Processing

Data is everywhere. Nowadays, it is most likely in digital form that can be processed by a machine. The data processing cycle is a series of operations on data, mainly including data collection, data input, data processing, data output. Data processing may include but is not limited to retrieval, transformation, and/or classification and hope to produce meaningful information. Figure 1 shows a generic example of a data processing system, or specifically a pattern recognition system.

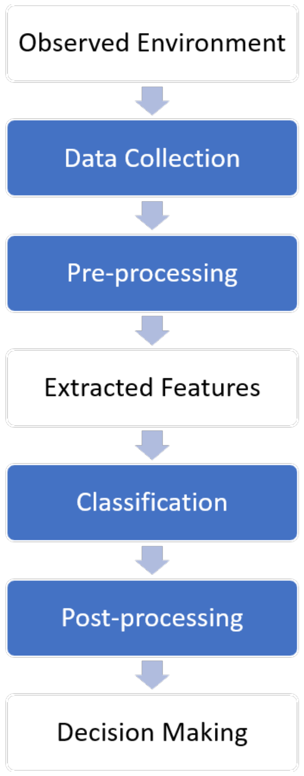

Figure 1: A data processing system

For each component in Figure 1, it may contain multiple sub-processes. First and foremost, what are the questions/problem sets that you would like to ask? Then, identify your environment. By observing environment, data can be collected and/or synthesized to simulate the environment. Collected data may or may not need to be cleaned depending on your problems and environment. Pre-processing may include cleaning data, generating primitive observations/features, ranking features, selecting features, extracting features, etc. Classification is the choice of model for machine learning or estimation. Post-processing includes actions for decision making, such as minimizing error rate or cost, evaluating classifiers, etc. All of these components may be constructed by data analysis, statistics, machine learning and their related techniques and methods.

Data repositories

1. UCI: https://archive.ics.uci.edu/ml/index.php and https://kdd.ics.uci.edu
2. Kaggle: https://www.kaggle.com/datasets
3. Amazon: https://registry.opendata.aws
4. Data.gov: https://www.data.gov

2 Understanding of Data (Data Types and Formats)

In this section, understanding the types and formats of data is fundamental to data science, AI, and data analytics. Knowledge in this area is essential not only for data preprocessing but also for selecting the appropriate algorithms for data analysis and model training. The types and format of data are described as raw data, such as account numbers, names, SSN, images, voice signals, videos, measurements, etc. This can also get even more complicated. Let's consider images that are jpeg, jpg, bmp, gif, tiff. In many cases, data can be the information held in headers of files, such as but not limited to, describing the name of the file, data format, compression type, location of the captured file, geographic location. The data in this case can be numeric (real, complex, positive) or symbolic (alphabet, ascii, words) in which values are [-10, -9, ...9, 10] or [easy, medium, difficult].

2.1 Data Types

Data types classify the characteristics of the data and decide what operations can be performed with them. They define the nature of the data, inform what kinds of operations can be performed, and guide the selection of statistical methods and machine learning algorithms. Understanding data types is crucial in fields like data science, AI, and data analytics. Some data types are described below:

Numerical Data

- Continuous: Data that can take any value within a range (e.g., height, weight).
- Discrete: Data that can take only integer values (e.g., the number of employees).

Categorical Data

- Nominal: Data without any natural order (e.g., colors, gender).
- Ordinal: Data with a natural order but the distance between categories is not uniform (e.g., ratings).

Textual Data

- Unstructured: Free-form text (e.g., reviews, comments).
- Structured: Formatted text (e.g., JSON, XML).

Time Series Data

- Data collected or recorded at a regular time interval (e.g., stock prices).
- Measurements recorded at different points in time. It's crucial for understanding trends, cycles, and patterns over time.

Image and Video Data

- Data in the form of images or videos, often used in computer vision tasks.

Audio Data

- Sound or voice data, usually utilized in natural language processing and machine learning models focused on audio analysis.

2.2 Data Formats

In addition, various data storage, programming languages, or algorithms may handle data differently due to data formats. There is no best format for all. Take flat files for example. The following table shows the same data in four different formats, which may result in using different syntax in different programming languages to read in different file formats.

Table 1: Data Formats

File Format	Example

Tagged file	BEGIN
	COUNTRY New Zealand
	NAME Tim Maoui
	END

CSV	country, name New Zealand, Tim Maoui

JSON	{
	"country": "New Zealand",
	"name": "Tim Maoui"
	}

Python	{
	'country': 'New Zealand',
	'name': 'Tim Maoui'
	}

2.3 Python Data Types

It should be noted that Python Data Types have built-in data types such as collections, dict, list set and tuple as described below:

- collections - The collections module provides a specialized container datatype tot he Python general purpose built-in containers. collections — Container datatypes
- dict - Dictionary is an unordered collection of key-value pairs. A mapping object maps hashable values to arbitrary objects. Mappings are mutable objects. There is currently only one standard mapping type, the dictionary. Mapping Types — dict
- list - List are mutable sequences, typically used to store collections of homogeneous items (where the precise degree of similarity will vary by application).. It is one of the most used datatype in Python and is very flexible. All the items in a list do not need to be of the same type. Lists
- set - A set object is an unordered collection of distinct hashable objects. Set is defined by values separated by comma inside braces . Items in a set are not ordered. Common uses include membership testing, removing duplicates from a sequence, and computing mathematical operations such as intersection, union, difference, and symmetric difference. Set Types — set, frozenset
- tuple - Tuples are immutable sequences, typically used to store collections of heterogeneous data (such as the 2-tuples produced by the enumerate() built-in). Tuples are also used for cases where an immutable sequence of homogeneous data is needed (such as allowing storage in a set or dict instance). Tuples once created cannot be modified. Tuples

3 Data Collection

Data collection is an essential phase in data science, AI, and analytics. It involves gathering information from various sources to be used for analysis. It enables one to answer relevant questions, evaluate outcomes, and make predictions about future possibilities. Several models and algorithms are utilized to automate and optimize the data collection process.

Sources of Data

- Primary Sources: Data collected directly by the researcher for a specific purpose. It includes surveys, observations, and experiments.

- Secondary Sources: Data collected by someone other than the user. Common sources include government publications, online databases, and internal records of organizations.

Considerations in Data Collection

- Ethics: Ensuring that data collection methods comply with ethical guidelines, particularly when dealing with sensitive or personal information.
- Quality: Ensuring that the data is accurate, consistent, and unbiased.
- Cost and Time: Consider the resources needed for data collection, including monetary costs and time investment.
- Privacy: Protecting the privacy of the individuals from whom the data is collected.
- Data Security: Ensuring that collected data is stored and handled securely.
- Sampling: If it is impossible to collect data from an entire population, a representative sample must be chosen.

Methods and techniques used in data collection

- Web Scraping Algorithms: Web scraping is the process of extracting data from websites. Several algorithms and libraries, such as Beautiful Soup, Scrapy, and Selenium, are used to automate this process. They can navigate through web pages, interact with forms, and extract the required information.
- Sensor Data Collection: In IoT (Internet of Things) applications, sensor data collection is crucial. Various algorithms are used to optimize energy consumption, accuracy and reliability, such as data aggregation algorithms and consensus algorithms.
- Stream Data Collection Algorithms: Real-time data streaming requires specific algorithms to handle continuous, fast-paced data inflow. Algorithms such as sliding-window algorithms are used to process and analyze data in real time.
- Data Integration Algorithms: Collecting data from multiple heterogeneous sources requires specialized algorithms to integrate and align the data. These may include data matching, data transformation, and entity resolution algorithms.
- Crowdsourcing Algorithms: Crowdsourcing involves collecting data from a large number of contributors, often through online platforms. Algorithms to ensure data quality, reward distribution, and task assignment are essential in this process.
- Social Media Mining Algorithms: Social media platforms are rich sources of data. Algorithms for sentiment analysis, community detection, trend analysis, etc. are commonly used to collect and process data from these platforms.

Data collection is a critical step in any research or analytical process. Different data collection models and algorithms are suited to different types of data, sources, and specific requirements of the analysis. Understanding these methods and selecting the appropriate approach is a vital skill, especially in fields like data science, AI, and analytics. It is essential to plan the data collection process meticulously to ensure that the data gathered will be suitable for the intended analysis. In the fields of AI research, data science, and data analytics, proper data collection can significantly impact the success of projects and the validity of the findings.

3.1 Data Mining

Data mining is a kind of data processing that contains knowledge discovery. It is to process raw data and gain meaningful and useful information at the end. The process includes looking for previously known or unknown patterns and/or anomalies in a larger scale. In addition, it may include predicting patterns. A large scale of processing is most likely expected to be methodological, systematic and automatic with acceptable response and error rates. It generally has large scale data sets with more complexed analysis and analytics involved.

Data Mining for Data Collection

Here, the focus is more on how data mining can assist in gathering valuable data for further analysis:

Web Scraping: Mining data from websites for specific information.

Social Media Mining: Extracting insights from social media platforms like Twitter, Facebook, and LinkedIn.

Text Mining: Gathering useful information from textual data like articles, reviews, and blogs.

Sensor Data Mining: Utilizing IoT devices to collect useful data.

Database Mining: Extracting data from databases by running specific queries that will give useful insights.

Stream Mining: Collecting and analyzing real-time data like stock prices or social media trends.

Spatial Data Mining: Collecting data with geographical or spatial information.

Data Mining using Python

Data mining for data collection in Python is often performed using libraries and frameworks designed to simplify the extraction, transformation, and loading of data. For Python the data mining tools that should be considered are listed as follows:

Beautiful Soup: Used for web scraping HTML and XML documents. It creates a parse tree that can be used to extract data easily.

Scrapy: A more robust and versatile web scraping framework that also allows you to scrape using APIs.

Pandas: Excellent for data manipulation and analysis, particularly useful for data cleaning and transformation.

Numpy/SciPy: Used for numerical operations, these libraries are especially helpful when the data mining tasks require mathematical computations.

NLTK and SpaCy: Libraries for natural language processing, particularly useful when mining textual data.

Sklearn: Provides simple and efficient tools for data mining and analysis, particularly useful for clustering, classification, and regression tasks.

NetworkX: Useful for the creation, manipulation, and study of the structure of complex networks.

TensorFlow/PyTorch: For more advanced, machine learning-based data mining tasks, like deep learning.

GeoPandas: Specialized for working with geospatial data.

4 Data Cleaning

Data cleaning, also known as data cleansing, is an essential step in data preprocessing, particularly in data analytics, data science, and AI. The process of data cleansing involves recognizing and rectifying inconsistencies or inaccuracies in data from a record set, table, or database. This includes recognizing incomplete, incorrect, or irrelevant parts of the data and then replacing, modifying, or deleting the coarse data, noisy data, outliers, and repetitive data to enhance its quality and efficiency. In some application areas of data science, data collection and data cleaning are critical to the entire analysis process. By doing so, it improves data quality and hopes to increase overall productivity and robustness along with reduce error rate. In addition, this step may take data scientists a very large potion of their time.

Here's an overview of the data cleaning process, techniques, and considerations:

Common Data Cleaning Tasks

- Handling Missing Values:
  - Removing rows or columns with missing values.
  - If class labels are associated with the data, impute data based on metrics from the associated class and not the entire data set.
  - Imputing missing values using techniques like mean, median, mode, or using algorithms like k-Nearest Neighbors.
- Removing Duplicates:
  - Identifying and removing duplicate rows that may have been recorded in error.
- Correcting Data Types:
  - Ensuring that each attribute is of the correct data type (e.g., numerical, categorical, date-time).
- Detecting and Handling Outliers:
  - Identifying extreme values that might be errors and handling them through techniques like clipping or transformation.
- Standardizing Values:
  - Making data consistent, such as converting all text to lowercase or using a standard unit of measurement. Methods such as normalization and standardization are described in the Data Transformation section.
- Error Correction:
  - Identifying and correcting human or system errors in data, such as typos or mislabeled classes. Using LLM can help automate this process.
- Normalization, Standardization, and Scaling:
  - Making data consistent, such as converting all text to lowercase or using a standard unit of measurement.
  - Scaling the numerical features to a standard range, like 0-1, to ensure that they are comparable.
  - Methods such as normalization and standardization are described in the Data Transformation section.

Reflection on data cleaning is essential. There are several factors to consider when cleaning data, such as accuracy, completeness, consistency, and uniformity. Accuracy refers to the degree to which the data is correct and reliable. Completeness is the extent to which the data contains all the necessary information. Consistency is the degree to which the data is consistent with other data sets. Uniformity is the extent to which the data is formatted in a consistent manner. All of these considerations are important when cleaning data.

- Understanding the Data: Before cleaning, it's essential to understand the data, its context, and what constitutes an error or inconsistency.
- Documentation: Keeping track of all the cleaning steps to ensure reproducibility.
- Ethics and Compliance: Ensuring that data cleaning doesn't violate privacy or other legal regulations.
- Data Verification: Assessing the quality of data on an ongoing basis during the cleaning process is known as data verification.

Data cleaning is an essential step in the data processing cycle, which can have a major effect on the precision and dependability of any following examination or modeling. It necessitates cautious arranging, execution, and approval. For Python data cleansing pandas and NumPy are great packages to use for various techniques in cleansing data. Seaborn is a great package for statistical data visualization to identify and analyze the various tools and techniques used.

5 Data Transforms

Data transformation is an integral part of data preprocessing in data analytics, data science, and AI. It involves converting data from its original form into a format that makes it more suitable for analysis or modeling. After features are generated it is necessary to preprocess the features that are to be used for classification. In many practical situations the classification model may receive input features whose values lie within different dynamic ranges. Thus, features with large values may inadvertently influence classification over features with small values. Another problem arises when a particular sample is not within the same area as the other features. Some of the procedures used for data preparation are feature standardization (Dillon and Goldstein, 1984, pp. 12-13) [1], feature min-max normalization (Theodoridis and Koutroumbas, 2006, pp. 214-215) [2], min-max global normalization (Guyon et al., 2006, pp. 254) [3], sigmoid normalization (Theodoridis and Koutroumbas, 2006, pp. 214-215) [2] and softmax scaling (Theodoridis and Koutroumbas, 2006, pp. 214-215) [2]. We use zero-mean normalization (feature standardization) and min-max normalization (feature normalization) and describe them in more details. The training vectors in this section are represented by $x_i = [x_1, x_2, \ldots, x_n] \in \mathbf{X}$ with a dimension of d and the number of samples defined as n.

5.1 Min-max normalization

The Min-max normalization performs a linear scaling on the original data. The normalization is calculated by estimates of the minimum and maximum of the values. The normalization technique is defined for the n available data samples and the k^{th} feature as:

$$\hat{x}_{ik} = \frac{x_{ik} - min(x_k)}{max(x_k) - min(x_k)}(b - a) + a, \qquad k = 1, 2, ..., l \text{ and } i = 1, 2, ..., n \tag{1}$$

where a and b are scaling factors. When a = 0 and b = 1 the individual feature values are in the range of [0,1]. In the event that the denominator of Equation (2.18) is equal to zero that feature is removed, avoiding the potential of normalizing a feature of constants.

Matlab Code

```
function [xStandard, indx] = dataNormalization(X,normType,decision)
% This code is for educational and research purposes of comparisons. This
% is code that provides the use of 4 data normalization methods.
%
% Input:
%   X [l x n] - l is the number of exemplars/samples/observation
%             - n is the dimension of the data
%   normType [1 x 1] - determines the type of normalization method used
%                normType = 1 - Global Normalization
%                                  X
%                           N = ---------
%                                 ||X||
%                normType = 2 - Local Normalization of the data.
%                                The data undergoes a line by line normalization
%                                with the Euclidean norm of the line. If it is
%                                desired to center the data, subtract the
%                                features mean before normalizing the features.
%                                By default the data should be centered.
%                                 Xi - mean(Xi)
%                           N = ---------------
%                                    ||Xi||
%
%                normType = 3 - Local Normalization of the data.
%
%                                                     Xi
%                                              N = ---------------
%                                                     ||Xi||
%
%                normType = 4 - Normalization of the data between [a, b].
%                                This method is the default and a = -1, b = 1
%                                 X - min(X)
%                (default)  N = ----------------- * (b-a)+a
%                                max(X)-min(X)
%
%   decision [1 x 1] - determines what to do in the event a standard
%                      deviation is equal to zero
%                decision = 1 - removes the features whos std dev = 0
%                decision = 2 - gives a fudge factor of meadian(s)
%                Default, decision = 1
```

```
%
% Output:
%   xNorm [l x n] - normalized data according to:
%   indx [1 x m] - where m is the number of features which have a std dev
%                  equal to zero. If no feature contins a std dev of zero
%                  the indx is returned as an empty matrix.
%
% Each feature was separately standardized, by substracting its mean and
% dividing by the standard deviation. To avoid dividing by zero, e.g. s = 0
% the feature is either removed if decision = 1, or the feature is devided
% by its mean if decision = 2. In the case of decision = 2 the mean is not
% substracted.
%
% Example:
%

if nargin < 2;normType = 4;decision = 1;end
if nargin < 3;decision = 1;end

[l, n] = size(X);
indx = [];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%          Global Normalization of the data.                                  %
%          Normalize the columns of Training data                             %
%                         X                                                   %
%                 N = --------                                                %
%                       ||X||                                                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if normType == 1
    NP = sqrt(diag(X'*X));
    xNorm = X./(ones(size(X,1),1)*NP');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%          Local Normalization of the data.                                   %
%                                                                             %
%                       Xi - mean(Xi)                                         %
%                 N = ---------------                                         %
%                         ||Xi||                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif normType == 2
    m = (mean(X));
    X = X - kron(ones(1,size(X,1)),m')';
    NP = sqrt(sum(X.^2,2));
    NP(find(NP == 0)) = 1;
    X = X./NP(:,ones(1,size(X,1)));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%          Local Normalization of the data.                                   %
```

```
93  %                                                                   %
94  %                             Xi                                    %
95  %                   N = ----------------                            %
96  %                           ||Xi||                                  %
97  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
98  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
99   elseif normType == 3
100      m = (mean(X));
101 
102      NP = sqrt(sum(X.^2,2));
103      NP(find(NP == 0)) = 1;
104      X = X./NP(:,ones(1,size(X,1)));
105 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
106 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
107 %          Normalization of the data between [a, b].                %
108 %                                                                   %
109 %                             X - min(X)                            %
110 %                    N = ----------------- * (b-a) + a              %
111 %                          max(X)-min(X)                            %
112 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
113 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
114  elseif normType == 4
115      Pmin = min(X);
116      Pmax = max(X);
117      a=-1; b=1;
118      minMax = Pmax - Pmin;
119      indx = find(minMax==0);
120      if indx% Ensures that a division by zero does not occur
121          Pmin(indx) = [];
122          Pmax(indx) = [];
123          X(:,indx) = [];
124      end
125      X = ((X - kron(ones(1,l),Pmin')')./...
126          (kron(ones(1,l),Pmax')' - kron(ones(1,l),Pmin')')).*(b-a)+ a;
127 end
128 xStandard = X;
```

Equation 1 is implemented in lines 114 - 127 in the provided Matlab code. The kron function is the Kronecker tensor product where Matlab provides the description as: K = kron(A,B) returns the Kronecker tensor product of matrices A and B. If A is an m-by-n matrix and B is a p-by-q matrix, then kron(A,B) is an m*p-by-n*q matrix formed by taking all possible products between the elements of A and the matrix B.

5.2 Z-score Normalization (Standardization)

The Z-score normalization is based on the mean and standard deviation of each feature. Each feature in this method is separately standardized by subtracting its mean and dividing by the standard deviation as follows:

$$\hat{x}_{ik} = \frac{x_{ik} - \mu_k}{\sigma_k}, \qquad k = 1, 2, ..., l \text{ and } i = 1, 2, ..., n \tag{2}$$

where μ_k and σ_k are defined as:

$$\mu_k = \frac{1}{n} \sum_{i=1}^{n} x_{ik} \tag{3}$$

$$\sigma_k = \left(\frac{1}{n-1} \sum_{i=1}^{n} \left(x_{ik} - \mu_k \right)^2 \right)^{1/2} \tag{4}$$

and m is the number of samples. In the event that the standard deviation of a particular feature is zero (e.g., each element of the observed feature is a constant value), the feature is discarded.

Note: That $\hat{x}_{ik}$ in Equation (1) and Equation (2) are not equal to each other, the two equations produce two different and unique normalized sets of values.

Matlab Code

```
function [xStandard, indx] = dataStandard(X,decision)
% This code is for educational and research purposes of comparisons. This
% is code that provides the use of the z-score normalization
% (standarization).
%
% Input:
%   X [l x n] - l is the number of exemplars/samples/observation
%             - n is the dimension of the data
%  decision [1 x 1] - determines what to do in the event a standard
%                     deviation is equal to zero
%           decision = 1 - removes the features whos std dev = 0
%           decision = 2 - if std dev = 0, then X/mean(X)
%           Default, decision = 1
%
% Output:
%   xStandard [l x n] - standardized data according to:
%                          X - mean(X)
%                     S = -------------
%                            std(X)
%  indx [1 x m] - where m is the number of features which have a std dev
%                 equal to zero. If no feature contins a std dev of zero
%                 the indx is returned as an empty matrix.
%
% Each feature was separately standardized, by substracting its mean and
% dividing by the standard deviation. To avoid dividing by zero, e.g. s = 0
% the feature is either removed if decision = 1, or the feature is devided
% by its mean if decision = 2. In the case of decision = 2 the mean is not
% substracted.
%
% Example:
%
```

```
32
33 if nargin < 2;decision = 1;end
34
35 [l, n] = size(X);
36
37 if decision == 1
38     s = std(X);
39     indx = find(s==0);
40     if indx
41         X(:,indx) = [];
42         s = std(X);
43         m = (mean(X));
44         StndP = X - kron(ones(1,l),m')';
45         xStandard = StndP./kron(ones(1,l),s')';
46     else
47         m = (mean(X));
48         StndP = X - kron(ones(1,l),m')';
49         xStandard = StndP./kron(ones(1,l),s')';
50     end
51 elseif decision == 2
52     s = std(X);
53     indx = find(s==0);
54     m = (mean(X));
55     if indx
56         s(indx) = max(m(indx));
57         m(indx) = 0;
58     end
59     StndP = X - kron(ones(1,l),m')';
60     xStandard = StndP./kron(ones(1,l),s')';
61 end
62
63 % In Matlab
64 % y = (x-xmean)*(ystd/xstd) + ymean;
65 % ymean - Mean value for each row of Y (default is 0)
66 % ystd - Standard deviation for each row of Y (default is 1)
```

Equation 2 is implemented in lines 33 - 61 in the provided Matlab code.

6 Feature Engineering

Feature engineering is a critical step in the machine learning pipeline that can greatly influence the performance of a model. It involves creating new features from the existing ones, selecting only those features that are most informative, or transforming features to a more suitable form. Here's an overview of feature engineering and its importance, especially in the fields of data science, AI, and data analytics.

What is Feature Engineering? Feature engineering is the process of using domain knowledge to generate features that make machine learning algorithms work. It's an art as much as a science, allowing the model to understand the data in a more nuanced way.

Importance of Feature Engineering:

- Improves Model Performance: Well-crafted features can enhance model accuracy, precision, recall, and other performance metrics.
- Reduces Computational Resources: By selecting only relevant features, the model can be trained more quickly and with less memory.
- Enhances Interpretability: Thoughtful feature engineering can make the model easier to understand and explain.
- Leverages Domain Knowledge: Incorporating expert knowledge into feature engineering can bridge the gap between raw data and the underlying patterns that a model needs to learn.

In [4] the focus was on the fusion of two modalities, visual signals represented by images and vocal signals which encode sounds, to form a multi-modal dataset to use in supervised classification tasks [5]. As shown in Figure 2 a framework for generating feature from raw data was presented. The raw image and vocal data is first pre-processed using discrete cosine transform (DCT), Discrete Fourier Transform (DFT), and wavelet decomposition using Daubechies wavelets. Subsequently, features are generated using Fisher's linear discriminant, eigenvalue decomposition, and first-order statistics. This allowed a set of features to be generated that would represent the two modalities for the ML models for classifying the 10 classes.

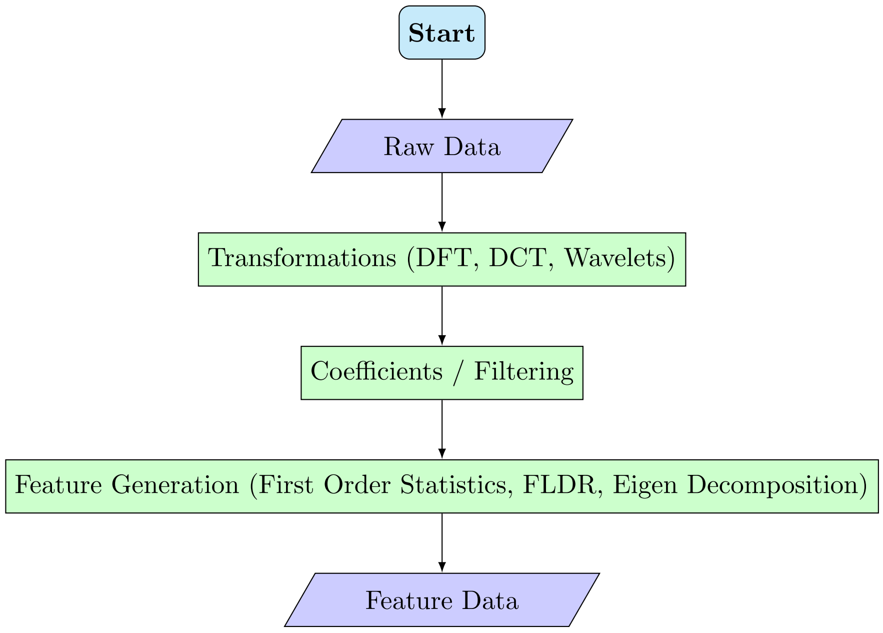
Start
Raw Data
Transformations (DFT, DCT, Wavelets)
Coefficients / Filtering
Feature Generation (First Order Statistics, FLDR, Eigen Decomposition)
Feature Data

Figure 2: Feature Engineering process flow to generate features.

Feature engineering is a sophisticated and essential step that requires both analytical skills and domain knowledge. It is a blend of creativity, experimentation, and understanding of machine learning algorithms. In teaching data science, AI, or data analytics, providing hands-on experience and insights into feature engineering can equip students and professionals with the tools to build more robust and effective models. Considering your teaching background in these fields, introducing real-world scenarios and examples in your curriculum can make the learning experience more engaging and applicable.

7 Outliers

Outliers are data points in a dataset that differ significantly from other observations. They can have a substantial impact on statistical analyses and machine learning models, potentially leading to misleading conclusions or poor predictions. Here is a closer look at outliers, how they can be identified, handled, and why they matter, especially in the fields of data analytics, data science, and AI.

Causes of Outliers:

- Mistakes in data entry can cause the formation of false outliers.
- Inaccurate readings can be caused by malfunctioning measurement devices or techniques. Mistakes in measurement can occur.
- In certain cases, outliers may be indicative of genuine and anticipated variations in the data.

Identifying Outliers:

Barnett and Lewis (1994) [6] offer a range of instructions to identify a few outliers. If there are a lot of outliers, the analyst must be cautious.Here are a few methods to identify the outliers:

- Scatter plots, histograms, and box plots can be used to visually detect outliers.
- Z-scores, Tukey's fences, and the Mahalanobis distance are all popular statistical techniques used to identify outliers.
- Algorithms such as Isolation Forest and One-Class SVM can be utilized to identify anomalies, particularly in data with a high number of dimensions.

Assuming that the outliers are errors, they can be removed. In certain cases, the outliers themselves may be of interest and can be examined independently. Two strategies to eliminate outliers are used when dealing with multivariate outliers. The first is a method in which the mean is used to set an upper and lower boundary of a confidence interval to identify an outlier and take out the sample (Barnett and Lewis, 1994) [6]. The second is a multivariate outlier technique proposed by Wilks (1963) [7]. The last method is the use of the Mahalanobis distance [8].

Outliers are an integral part of data analysis and preprocessing. It is essential to comprehend, recognize, and treat outliers correctly to obtain reliable and precise results. Outliers can distort the results and reduce the precision of models. Outliers can influence the comprehension of data patterns and associations. Managing outliers correctly can make models more resilient to changes in the data.

7.1 Confidence Interval Outlier Removal

In confidence interval outlier removal, any sample outside of the confidence interval is considered an outlier. This method assumes the data is normally distributed and generates a confidence interval for each feature. The first step identifies an upper and lower limit means from the global mean as follows:

$$\mu_{upper} = \frac{1}{S_{upper}} \sum_{i \in S_{upper}} x_i, \qquad for \quad x_i > \mu \tag{5}$$

$$\mu_{lower} = \frac{1}{S_{lower}} \sum_{i \in S_{lower}} x_i, \qquad for \quad x_i < \mu \tag{6}$$

where μ is the global mean vector

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{7}$$

where m is the number of samples, Supper and Slower are the number of samples meeting the criteria $x_i > \mu$ and $x_i < \mu$, respectively. The term $i \in S_{lower}$ and $i \in S_{upper}$ indicate the indices when the criteria $x_i < \mu$ and $x_i > \mu$ are met. This now leads to the confidence interval defined as:

$$[\mu_{lower} - (\alpha(\mu - \mu_{lower})), \mu_{upper} + (\alpha(\mu_{upper} - \mu))] \tag{8}$$

where m is the parameter set by the user. A good starting point is $\alpha = 0.5$ allowing the parameter to be adjusted based on the data set being analyzed. The terms multiplied by α in the above equation can be replaced by the critical of the t- distribution as described by Barnett and Lewis (1994, page 74) providing robustness of validity for the confidence interval. Another alternate modification to the above equation is to simply replace $(\mu - \mu_{lower})$ by the standard deviation of μ_{lower} and $(\mu_{upper} - \mu)$ by the standard deviation of μ_{upper} allowing the standard deviation to determine the confidence interval.

7.2 Wilks' Outlier Removal

Wilks' outlier removal technique uses an upper bound for detection of a single outlier from a set of normal multivariate samples in which the maximum squared Mahalanobis distance (as shown in the equation below) approaches an F distribution (Wilks, 1963) [7].

$$Mah\_Dist_i^2 = (x_i - \mu)\Sigma^{-1}(x_i - \mu)^T \tag{9}$$

In multivariate outlier detection the normality between samples is assessed. A partial mathematical description is provided by Rencher (2002, pp. 101-104) [9] and expanded in application by Trujillo-Ortiz, et al. (2008).
Determining the threshold is defined by the F distribution critical value (inverse of F cumulative distribution function) with n and $(m - n - 1)$ degrees of freedom using the Bonferroni correction (Bonferroni, 1935; 1936). The final critical value is defined by:

$$C_v = \frac{D(N-1)^2}{N(N-D-1) + (lnF)} \tag{10}$$

The index of an outlier(s) is identified by the following criteria:

$$MD_i^2 \geq C_v \tag{11}$$

Fc = finv(1-a/n,p,n-p-1);
F distribution critical value with p and n-p-1 degrees of freedom using the Bonferroni correction.
$ACR = (p(n-1)^2 Fc)/(n(n-p-1) + (npFc))$

In python from Junqi Chen, DS student from Algorithms for Data Science Spring 2021:

This method is provided in full detail by Trujillo-Ortiz, et al. (2008).

Matlab Code

```
function wilksMultivariateOutlier(X,alpha)
% MOUTLIER1 Detection of Outlier in Multivariate Samples Test.
% This test is based on the Wilks'method (1963) designed for detection of a
% single outlier from a normal multivariate sample and approaching the
% maximun squared Mahalanobis distance to a F distribution function by the
% Yang and Lee (1987) formulation. A significative squared Mahalanobis
% distance means an outlier. To test the outlier, this function calls the
% ACR function.
%
% Syntax: function wilksMultivariateOutlier(X,alpha)
%
% Inputs:
%      X - multivariate data matrix.
%      alpha - significance level (default = 0.05).
%
% Output:
%      - Table of outliers detected in a multivariate sample.
%
% Additional Files Needed:
%     ACR.m
%
% Example: From the example of Rencher (2002, p. 79). Consisting of
% measurements of the ramus bone length at four different ages
% (8, 8.5, 9 & 9.5 yr) on each of 20 boys. We are interesting to detect any
% multivariate outliers. We use a significance of 0.1.
%
%                     -----------------------------------------------
%                                             Age
%                              ----------------------------------
%                      Subject    8          8.5         9          9.5
%                     -----------------------------------------------
%                          1     47.8        48.8        49.0        49.7
%                          2     46.4        47.3        47.7        48.4
%                          3     46.3        46.8        47.8        48.5
%                          4     45.1        45.3        46.1        47.2
%                          5     47.6        48.5        48.9        49.3
%                          6     52.5        53.2        53.3        53.7
%                          7     51.2        53.0        54.3        54.4
%                          8     49.8        50.0        50.3        52.7
%                          9     48.1        50.8        52.3        54.4
%                         10     45.0        47.0        47.3        48.3
%                         11     51.2        51.4        51.6        51.9
%                         12     48.5        49.2        53.0        55.5
%                         13     52.1        52.8        53.7        55.0
%                         14     48.2        48.9        49.3        49.8
%                         15     49.6        50.4        51.2        51.8
%                         16     50.7        51.7        52.7        53.3
%                         17     47.2        47.7        48.4        49.5
%                         18     53.3        54.6        55.1        55.3
%                         19     46.2        47.5        48.1        48.4
%                         20     46.3        47.6        51.3        51.8
%                  ------------------------------------------------------------
%
% Total data matrix must be:
%     X = [47.8 48.8 49 49.7;
%          46.4 47.3 47.7 48.4;
%          46.3 46.8 47.8 48.5;
%          45.1 45.3 46.1 47.2;
%          47.6 48.5 48.9 49.3;
%          52.5 53.2 53.3 53.7;
%          51.2 53 54.3 54.4;
%          49.8 50 50.3 52.7;
%          48.1 50.8 52.3 54.4;
%          45 47 47.3 48.3;
%          51.2 51.4 51.6 51.9;
%          48.5 49.2 53 55.5;
%          52.1 52.8 53.7 55;
%          48.2 48.9 49.3 49.8;
%          49.6 50.4 51.2 51.8;
%          50.7 51.7 52.7 53.3;
%          47.2 47.7 48.4 49.5;
%          53.3 54.6 55.1 55.3;
%          46.2 47.5 48.1 48.4;
%          46.3 47.6 51.3 51.8];
%
% Calling on Matlab the function:
%                 wilksMultivariateOutlier(X,0.10)
%
% Answer is:
%
% Table of observation(s) resulting as multivariate outlier(s).
% ------------------------------------------------------
%                             D2
% Observation              observed
% ------------------------------------------------------
%       9                  11.0301
% ------------------------------------------------------
% With a given significance level of: 0.10
% Critical value for the maximum squared Mahalanobis distance: 10.9645
% D2 = squared Mahalanobis distance.
%
% Created by A. Trujillo-Ortiz, R. Hernandez-Walls, A. Castro-Perez and
%            K. Barba-Rojo
%            Facultad de Ciencias Marinas
%            Universidad Autonoma de Baja California
%            Apdo. Postal 453
%            Ensenada, Baja California
%            Mexico.
%            atrujo@uabc.mx
%
% Copyright. September 13, 2006.
%
%  To cite this file, this would be an appropriate format:
%  Trujillo-Ortiz, A., R. Hernandez-Walls, A. Castro-Perez and
%    K. Barba-Rojo. (2006).
%    MOUTLIER1:Detection of Outlier in Multivariate Samples Test. A MATLAB
%     file. [WWW document].
%    URL http://www.mathworks.com/matlabcentral/fileexchange/
%        loadFile.do?objectId=12252
%
% References:
% Rencher, A. C. (2002), Methods of Multivariate Analysis. 2nd. ed.
%           New-Jersey:John Wiley & Sons. Chapter 4, pp. 101-104.
% Wilks, S. S. (1963), Multivariate Statistical Outliers. Sankhya,
%           Series A, 25: 407-426.
% Yang, S. S. and Lee, Y. (1987), Identification of a Multivariate
%           Outlier. Presented at the Annual  Meeting of the American
%           Statistical Association, San Francisco, August 1987.
%

if nargin < 2,
   alpha = 0.05;  %(default)
end

if nargin < 1,
   error('Requires at least one input arguments.');
end

mX = mean(X); %Means vector from data matrix X.
[n,p] = size(X);
difT = [];

for j = 1:p;
   eval(['difT=[difT,(X(:,j)-mean(X(:,j)))];']); %squared Mahalanobis dist.
end

S = cov(X);
D2T = difT*inv(S)*difT';
[D2,cc] = sort(diag(D2T));  %Ascending squared Mahalanobis distances.

D2C = ACR(p,n,alpha);

idx = find(D2 >= D2C);
o = cc(idx);
io = D2(idx);

if isempty(o);
    disp(' ')
    fprintf('With a given significance level of: %.2f\n', alpha);
    disp('Non observation(s) resulting as multivariate outlier(s).');
else
    disp(' ')
    disp('Table of observation(s) resulting as multivariate outlier(s).')
    fprintf('------------------------------------------------\n');
    disp('                              D2');
    disp('Observation               observed');
    fprintf('------------------------------------------------\n');
    fprintf(' %6.0f              %10.4f\n',[o,io].');
    fprintf('-----------------------------------------------\n');
    fprintf('With a given significance level of: %.2f\n', alpha);
    fprintf('Critical value for the maximum squared Mahalanobis')
    fprintf(' distance: %.4f\n', D2C);
    disp('D2 = squared Mahalanobis distance.');
end

return,

function x = ACR(p,n,alpha);
%ACR Upper percentiles critical value for test of single multivariate
% normal outlier. From the method given by Wilks (1963) and approaching to
% a F distribution function by the Yang and Lee (1987) formulation, we
% provide an m-file to get the critical value of the maximun squared
% Mahalanobis dist. to detect outliers from a normal multivariate sample.
%
% Syntax: function x = ACR(p,n,alpha)
%   $$ The function's name is giving as a gratefull to Dr. Alvin C. Rencher
%      for his unvaluable contribution to multivariate statistics with his
%      text 'Methods of Multivariate Analysis'.$$
%
%     Inputs:
%          p - number of independent variables.
%          n - sample size.
%      alpha - significance level (default = 0.05).
%
%     Output:
%          x - critical value of the maximun squared Mahalanobis distance.
%
% We can generate all the critical values of the maximun squared
% Mahalanobis distance presented on the Table XXXII of by Barnett and
% Lewis (1978) and Table A.6 of Rencher (2002). Also with any given
% significance level (alpha).
%
% Example: For p = 3; n = 25; alpha=0.01;
%
% Calling on Matlab the function:
%                 ACR(p,n,alpha)
%
% Answer is:
%
%     13.1753
%
%  Created by A. Trujillo-Ortiz, R. Hernandez-Walls, A. Castro-Perez and
%             K. Barba-Rojo
%             Facultad de Ciencias Marinas
%             Universidad Autonoma de Baja California
%             Apdo. Postal 453
%             Ensenada, Baja California
%             Mexico.
%             atrujo@uabc.mx
%
```

```
211 % Copyright. August 20, 2006.
212 %
213 % To cite this file, this would be an appropriate format:
214 % Trujillo-Ortiz, A., R. Hernandez-Walls, A. Castro-Perez and
215 %   K. Barba-Rojo. (2006).
216 %   ACR:Upper percentiles critical value for test of single multivariate
217 %   normal outlier. A MATLAB file. [WWW document].
218 %   URL http://www.mathworks.com/matlabcentral/
219 %   fileexchange/loadFile.do?objectId=12161
220 %
221 % References:
222 % Barnett, V. and Lewis, T. (1978), Outliers on Statistical Data.
223 %           New-York:John Wiley & Sons.
224 % Rencher, A. C. (2002), Methods of Multivariate Analysis. 2nd. ed.
225 %            New-Jersey:John Wiley & Sons. Chapter 13 (pp. 408-450).
226 % Wilks, S. S. (1963), Multivariate Statistical Outliers. Sankhya,
227 %            Series A, 25: 407-426.
228 % Yang, S. S. and Lee, Y. (1987), Identification of a Multivariate
229 %            Outlier. Presented at the Annual  Meeting of the American
230 %            Statistical Association, San Francisco, August 1987.
231 %
232
233 if nargin < 3,
234    alpha = 0.05; %(default)
235 end;
236
237 if (alpha <= 0 | alpha >= 1)
238    fprintf('Warning: significance level must be between 0 and 1\n');
239    return;
240 end;
241
242 if nargin < 2,
243    error('Requires at least two input arguments.');
244    return;
245 end;
246
247 a = alpha;
248 Fc = finv(1-a/n,p,n-p-1); %F distribution critical value with p and n-p-1
249                           % degrees of freedom using the Bonferroni correction.
250 ACR = (p*(n-1)^2*Fc)/(n*(n-p-1)+(n*p*Fc));
251   % = ((-1*((1/(1+(Fc*p/(n-p-1))))-1))*((n-1)^2))/n;
252 x = ACR;
253
254 return,
```

It should be noted that the Mahalanobis distance is given on Equation 9 and implemented on lines 133-138; the Bonferroni correction is given in Equation 10 and implemented on lines 247 - 250; and the decision criteria shown in Equation 11 is implemented in line 143.

7.3 Mahalanobis Distance for Outlier Removal

Here the Mahalanobis distance is calculated for each observation within a class using the class mean μ by feature and the class inverse covariance matrix Σ^{-1} [8]. Note that the following equation only differs from Equation 9 in that the square root is taken on each side of the equation as shown below.

$$D_i = \left((x_i - \mu)\Sigma^{-1}(x_i - \mu)^T\right)^{1/2} \tag{12}$$

In Figure 7.3 the Setosa flower type (species) is shown in blue, Versicolor flower type (species) is shown in red, and the Virginica flower type (species) is shown in green. The x-axis is the sepal length and the y-axis is the petal width for this figure.

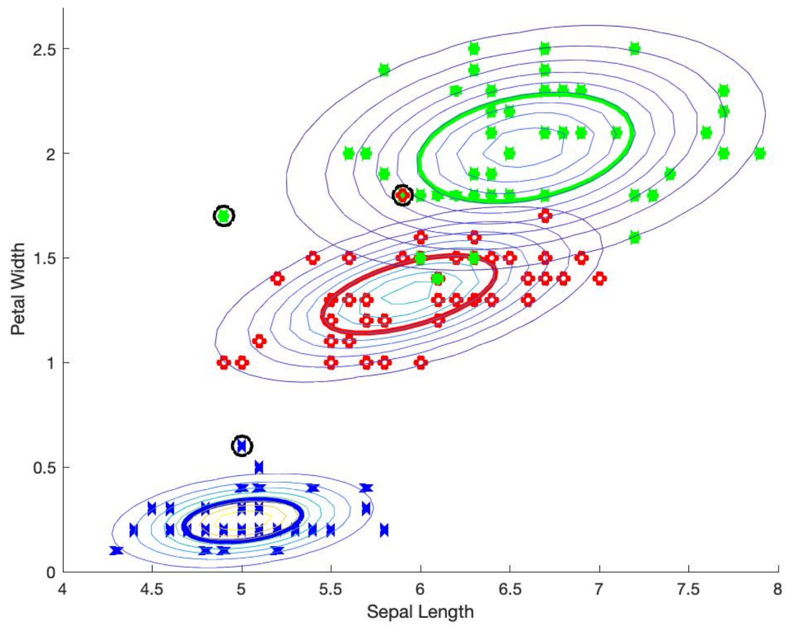

Figure 3: Outlier Identification - Here the Iris data by flower type has the observation with the largest observation identified based on the Mahalanobis distance.

The Mahalanobis distance can be used to identify the observations furthest from the mean by feature while accounting for the variance of each feature and the rotation of the data with the use of the covariance matrix. In this approach, the values returned from the Mahalanobis distance are sorted in descending with the value with the largest distance analyzed as a potential outlier. It should be apparent that the three observations for each class circled in black appear to be identified as outliers. These observations can be removed with the potential benefit of having a tighter standard deviation of the data being analyzed.

Matlab Code

```
function dist = mahalan(X,Mean,Cov)
% MAHALAN Computes Mahalanobis distance.
%
% Synopsis:
%  dist = mahalan(X,Mean,Cov)
%
% Description:
%  It computes Mahalanobis distance between column vectors
%  of matrix X and vector Mean with matrix Cov, i.e.,
%
%     dist(i) = (X(:,i)-Mean)'*inv(C)*(X(:,i)-Mean)
%
```

```
13 %  for all i=1:size(X,2).
14 %
15 % Input:
16 %  X [dim x num_data] Input data.
17 %  Mean [dim x 1] Vector.
18 %  Cov [dim x dim] Matrix.
19 %
20 % Output:
21 %  dist [1 x num_data] Mahalanobis distance.
22 %
23 % Example:
24 %  It plots isolines of Mahalanobis distance.
25 %
26 %  [Ax,Ay] = meshgrid(linspace(-5,5,100), linspace(-5,5,100));
27 %  dist = mahalan([Ax(:)';Ay(:)'],[0;0],[1 0.5; 0.5 1]);
28 %  figure; contour( Ax, Ay, reshape(dist,100,100));
29 %
30
31 % About: Statistical Pattern Recognition Toolbox
32 % (C) 1999-2003, Written by Vojtech Franc and Vaclav Hlavac
33 % <a href="http://www.cvut.cz">Czech Technical University Prague</a>
34 % <a href="http://www.feld.cvut.cz">Faculty of Electrical Engineering</a>
35 % <a href="http://cmp.felk.cvut.cz">Center for Machine Perception</a>
36
37 % Modifications:
38 % 28-apr-2004, VF
39
40 [dim, num_data] = size( X );
41
42 XC = X - repmat(Mean,1,num_data);
43 dist= sum((XC'*inv( Cov ).*XC')',1);
44
45 return;
```

It should be noted that the Mahalanobis distance is given on Equation 12 and implemented on lines 42 and 43.

8 Feature Ranking and Selection

The major task in feature selection, given a large number of features, is to select the most important features and reduce the dimensionality while retaining class discriminatory information. This procedure is important when determining which features are to be used to train the classification model. If features with little discrimination power are selected the subsequent classification model will lead to poor classification performance. On the other hand, if information rich features are selected the design of the classifier can be greatly simplified. In a more quantitative description, feature selection leads to large between-class distances and small within-class distances in the feature space. That is, features should separate different classes by a large distance, and should have small distance values between objects in the same class. Several methods are available to identify individual features with linear separation, a few ranking and selection methods include; divergence measure (Fukunaga, 1990; Theodoridis and Koutroumbas, 2006) [10] [2], Bhattacharyya distance (Bhattacharyya, 1943; Fukunaga, 1990) [11] [10] and Fisher's linear discriminant ratio (Fisher, 1936; 1943; Dillon and Goldstein, 1984; Fukunaga, 1990; van der Heijden et al., 2004; Bishop, 1995, 2006; Theodoridis and Koutroumbas, 2006). [12] [1] [10] [13] [14] [15] [2]

When measuring nonlinear class separability, care must be taken when using feature ranking methods. Ranking methods developed for specific classifiers are often best suited for determining the best set of ranked features. For neural network classifiers features are ranking and selected based on a saliency metric (Ruck et al., 1990; Belue and Bauer, 1995) [16] [17] and signal-to-noise ratio (Bauer et al., 2000). For kernel based classifiers, such as kernel

Fisher's discriminant and support vector machines, method-specific techniques are best suited for ranking. These techniques include recursive feature elimination (Guyon et al., 2002; Guyon, 2007) [3] [18], zero-norm feature ranking (Weston et al., 2003) [19], gradient calculations using recursive feature elimination (Rakotomamonjy, 2003) [20], and kernel Fisher's discriminant using recursive feature elimination (Louw and Steel, 2006). [21]

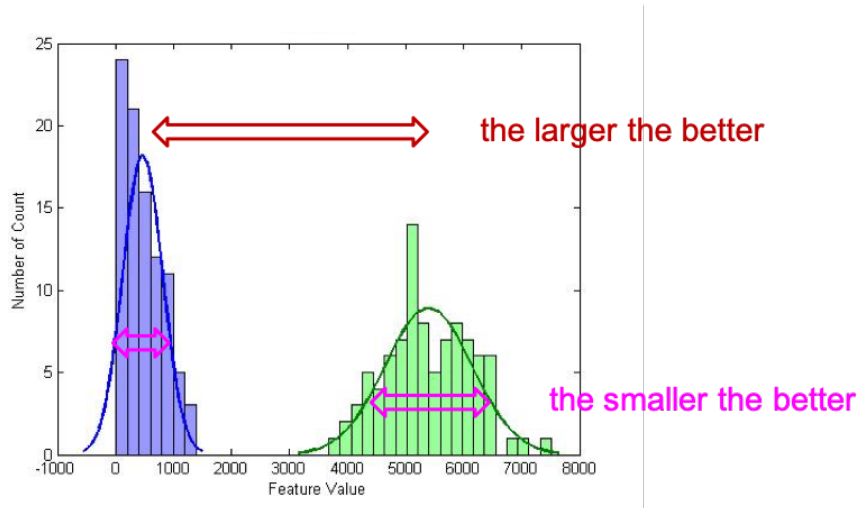

Figure 4: Feature Selection - ranking method based on between class means and within class variances.

Features should be ranked in the space that the classifier will be used. Features should be ranked individually with a metric that give a value on how well the feature classifies the observations. Once each feature is ranked an iterative method should be used to select the first two features to identify the well the top two features classify. Next select the top three features to identify how well the features classify. Continue this process until the classification accuracy fall or reaches a steady state.
Now let's see how you would potentially use the mean and standard deviation for the Iris data set, you would take the three mean and std values (m1, m2, m3, s1, s2, s3) for each class for a specific feature. You would then feed in observations 1 (x1) and use something like the Mahalanobis distance to determine which class x1 is closest to as follows: d1 = (x1 - m1)*s1*(x1-m1) d2 = (x1 - m2)*s2*(x1-m2) d3 = (x1 - m3)*s3*(x1-m3) You would assign x1 to the class with the smallest d = [d1, d2, d3] value. You would continue this process for the other 149 observation for feature 1. Compare how well the observations were classified to the original assigned classes. Save that value as the value for feature 1.
Do this process for feature2, feature 3, feature 4.
The feature with the largest classification is your top feature, the feature with the second highest classification is the second ranked feature and so on.
I would then follow this process by taking the mean and covariance of the top two features followed by calculating again the Mahalanobis distance to determine how well the top two features classify. Then do this for the top three features and so on to determine how well the sets of features separate the data.

8.1 Bhattacharyya Distance

The Bhattacharyya distance is used as a class separability measure. For two-class normal distributions the Bhattacharyya distance is defined as:

$$B = \frac{1}{8}(\mu_{-1} - \mu_{+1})^T \left(\frac{\Sigma_{-1} + \Sigma_{+1}}{2}\right)^{-1} (\mu_{-1} - \mu_{+1}) + \frac{1}{2} ln\left(\frac{||\frac{\Sigma_{-1}+\Sigma_{+1}}{2}||}{\sqrt{||\Sigma_{-1}||||\Sigma_{+1}||}}\right) \tag{13}$$

where || denotes the determinant of the respective matrix. The Bhattacharyya distance corresponds to the optimum Chernoff bound when $\Sigma_{-1} = \Sigma_{+1}$. It is readily seen that in this case the Bhattacharya distance becomes proportional to the Mahalanobis distance between the means. It should be noted that the Bhattacharya distance consists of two terms. The first term gives the class separability due to the mean difference and disappears when $\mu_{-1} = \mu_{+1}$. The second term gives the class separability due to the covariance difference and disappears when $\Sigma_{-1} = \Sigma_{+1}$ (Fukunaga, 1990). [10]

The Bhattacharyya distance for the multi-class case is represented as:

$$B_{ij} = \frac{1}{8}(\mu_i - \mu_j)^T \left(\frac{\sigma_i + \sigma_j}{2}\right)^{-1} (\mu_i - \mu_j) + \frac{1}{2} ln\left(\frac{\sigma_i^2 + \sigma_j^2}{2\sigma_i \sigma_j}\right), \qquad for \quad i \neq j \tag{14}$$

where $i, j \in \mathbb{Z}$ in this case corresponding to the classes $C = C_j = [C_1, C_2, \ldots, C_c], j = 1, 2, \ldots, c$. In this case for each feature an individual class is compared to the remaining classes based on distance. The features are assigned a ranking value according to the greatest distance between classes.

8.2 Fisher's Linear Discriminant Ratio (FDR/F-Score)

The FDR is used to quantify the separability capabilities of individual features (Fisher, 1936) [12]. FDR is a simple technique which measures the discrimination of sets of real numbers. The within-class scatter matrix is defined as

$$S_W = \sum_C P_C S_C \tag{15}$$

where S_c is the covariance matrix for class $\mathbf{C} \in -1, +1$

$$S_W = \sum_{\substack{i=1, \\ i \in C}}^{l_C} (\mathbf{x} - \mu_C)(\mathbf{x} - \mu_C)^T \tag{16}$$

and P_C is the *a priori* probability of class $\mathbf{C}$. That is, $P_C \approx \lambda_C / \lambda$, where λ_C is the number of samples in class $\mathbf{C}$, out of a total of λ samples. The between-class scatter matrix is defined as

$$S_B = \sum_{\mathbf{C}} (\mu - \mu_C)(\mu - \mu_C)^T \tag{17}$$

where μ is the global mean vector

$$\mu = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i \tag{18}$$

and the class mean vector μ_C is defined as

$$\mu_C = \frac{1}{n_C} \sum_{\substack{i=1, \\ i \in C}}^{n_C} \mathbf{x}_i \tag{19}$$

These criteria take a special form in the one-dimensional, two-class problem. In this case, it is easy to see that for equiprobable classes $|S_W|$ is proportional to $\sigma_i^2 + \sigma_j^2$ and $|S_B|$ proportional to $(\mu_{-1} - \mu_{+1})^2$. Combining SB and SW, the Fisher's Discriminant ratio results in the following equation

$$\boldsymbol{FDR} = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2} \tag{20}$$

FDR is sometimes used to quantify the separability capabilities of individual features. For the multi-class case, averaging forms of FDR can be used. One possibility is

$$\boldsymbol{FDR} = \sum_{i}^{M} \sum_{j \neq i}^{M} \frac{(\mu_i - \mu_j)^2}{\sigma_i^2 + \sigma_j^2} \tag{21}$$

where the subscripts i, j refer to the mean and variance corresponding to the feature under investigation for the classes C_i, C_j, respectively. For the one-dimensional multi-class case, the Fisher's discriminant ratio is modified as:

$$FDR_{ij} = \frac{(\mu_i - \mu_j)^2}{\sigma_i^2 + \sigma_j^2} \tag{22}$$

Python Code

```python
from itertools import combinations
import numpy as np
import pandas as pd

df = pd.read_csv("iris.csv")
y = df.species.astype("category").cat.codes.values
X = np.array(df.iloc[:, 0:4])
print(X.shape)
def FisherMultiClassFeatureRanking(X, y, method):
    Classes = np.unique(y)  # number of classes
    ranking = {}  # result dictionary

    def get_FDR(X, y, Classes):
        """
        Calculates Fisher's Linear Discriminant Ratio between two classes
        X: [num_data x Dim] training vectors.
        y: [num_data x 1] labels (class) of training data.
        method: Only for the multi-class case. This is how the scores of the
            different
        one-vs-rest fisher's score are conbined:
            1 = min
            2 = max
            3 = sum
            4 = average - (min+max)/2
        """
        X_1, X_2 = X[np.where(y == Classes[0])], X[np.where(y == Classes[1])]
        mu_1 = np.mean(X_1, axis=0)
        mu_2 = np.mean(X_2, axis=0)
        s_1_sq = np.std(X_1, axis=0) ** 2
        s_2_sq = np.std(X_2, axis=0) ** 2
        FDR = (mu_1 - mu_2) ** 2 / (s_1_sq + s_2_sq)
        return FDR

    if len(Classes) < 3:  # if two-class
        fdr = get_FDR(X, y, Classes)
        for idx, x in enumerate(np.argsort(fdr)[::-1]): # sorting in descending
            order
            ranking[f"feature_{x+1}"] = idx + 1
    if len(Classes) >= 3:  # multiclass
        combs = list(
            combinations(Classes, 2)
        )  # a list of combinations of classes for calculation
        arr = np.zeros(
            (
                len(combs),
                X.shape[1],
            )
        )
        for idx, i in enumerate(combs):
            c = list(i)
            arr[idx, :] = get_FDR(X, y, c)

        def avg_minus_max_min(arr, axis=0):
            return (
                np.mean(arr, axis=axis)
                - (np.max(arr, axis=axis) + np.min(arr, axis=axis)) / 2
            )

        methods = {
            1: np.min,
            2: np.max,
            3: np.sum,
            4: avg_minus_max_min,
        }  # four possible methods
        fdr = methods[method](arr, axis=0)
        for idx, x in enumerate(np.argsort(fdr)[::-1]): # sorting in descending
    order
            ranking[f"feature_{x+1}"] = idx + 1
    return ranking
```

8.3 Signal-to-Noise Feature Selection

One method for neural networks feature selection uses a signal-to-noise ratio (SNR) saliency measure (Bauer et al., 2000) . This measure directly compares the saliency of a feature to that of an injected noise feature. The SNR saliency measure is computed using the following:

$$SNR_i = 10\log_{10}\left(\frac{\sum_{j=1}^{J}(w_{i,j}^1)^2}{\sum_{j=1}^{J}(w_{N,j}^1)^2}\right) \tag{23}$$

where SNR_i is the value of the SNR saliency measure for feature i, J is the number of hidden nodes, $w_{i,j}^1$ is the first layer weight from node i to node j, and $w_{N,j}^1$ is the first layer weight from the injected noise node N to node j. The weights, for the noise feature are initialized and updated in the same fashion as the weights, emanating from the other features in the first layer. The injected noise feature is created such that its distribution follows that of a Uniform $(0,1)$ random variable. The SNR screening method potentially requires only a single training run, because the SNR saliency measure appears highly robust relative to the effects of weight initialization. For the classification method probabilistic neural network described in the classification section, this method is used to determine the appropriate subset of features.

9 Dimensionality Reduction

Another approach to reducing the dimension of the input features is to use a transformed space instead of the original feature space. For example using a transformation $f(x)$ that maps the data points $\mathbf{x}$ of the input space, $x[n]$, into a reduced dimensional space $x'[p]$, where $n > p$, creates features in a new space that may have better discriminatory properties. Classification is based on the new feature space rather than the input feature space. The advantage of feature extraction with the use of dimensionality reduction as described here over feature selection is that no information from any of the elements of the measurement vector is removed. In some situations feature extraction is easier than feature selection. A disadvantage of feature extraction is that it requires the determination of a suitable transformation $f(x)$. Some methods include principal component analysis (Hotelling, 1933; Dillon and

Goldstein, 1984) [22] [1] and kernel principal component analysis (Scholkopf et al., 1998; Bishop, 2006) [23] [15]. If the transformation chosen is too complex, the ability to generalize from a small data set will be poor. On the other hand, if the transformation chosen is too simple, it may constrain the decision boundaries to a form that is inappropriate to discriminate between classes. Another disadvantage is that all features are used, even if some of them have noise like characteristics. This might be unnecessarily expensive in term of computation (van der Heijden et al., 2004) [13]. It should be noted that the transformation used for the input features in the training of the classification model should also be used for the testing features.

In this section the reduction in dimensionality can also be accomplished using PCA. How ever PCA alone may not be sufficient, it may require the used of factor analysis in which the factors are rotated in order to group the appropriate variables based on their correlations. Examples will be given, however the exact details are outside of the scope of this document. In the Machine Learning I module linear discriminant analysis will be described which will describe dimensionality reduction in more detail using PCA.

9.1 Principal Component Analysis (PCA)

In PCA, there are assumptions made in the derivation that must be accounted for when scaling of the variables and the applicability. We will see later that PCA can be used for dimensionality reduction, however, caution should be used to ensure information representing the data is preserved in the principal components.

9.1.1 Principal Component Analysis (PCA)

The idea of feature extraction using PCA (Hotelling, 1933) [22] is to represent a new space in a way to extract mutually uncorrelated features from the current space. The new features are known as the principal components after transform mapping. The dimensionality assessment is accomplished by extracting the principal components from the correlation matrix and retaining only the factors described in Kaiser's criterion (eigenvalues: $\lambda \geq 1$) (Kaiser, 1960) [24]. The criterion is used as a guide line to determine the number of principal components to retain by calculating the correlation matrix of the input features. Each observed variable contributes one unit of variance to the total variance in the data set. Hence, any principal component that has an eigenvalue, λ greater than one accounts for a greater amount of variance than had been contributed by one variable. Additionally, a principal component that displays an eigenvalue less than one indicates less variance than had been contributed by one variable. The covariance matrix, Σ, is used to extract eigenvectors, e, retaining only the number of principal components corresponding to Kaiser's criterion.

The basic concept of feature extraction using PCA is to map x onto a new space capable of reducing the dimensionality of the input space. The data is partitioned by variance using a linear combination of 'original' factors. To perform PCA, let $\mathbf{x} = [x_1, x_2, \ldots, x_n] \in X_n$ be a set of training vectors from the n-dimensional input space X_n. The set of vectors $\hat{\mathbf{x}} = [\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m] \in \hat{X}_m$ is a lower dimensional representation of the input training vectors $\mathbf{x}$ in the m-dimensional space $\hat{X}_m$. The vectors $\hat{x}$ are obtained by the linear orthonormal projection

$$\hat{\mathbf{x}} = \mathbf{A}^T(\mathbf{x} - \mu) \tag{24}$$

where $\mathbf{A}$ is an $[n \times m]$ matrix containing the top m eigenvectors and μ is the mean of the each set of features from $\mathbf{x}$.

To find the eigenvectors and eigenvalues of a dataset begin with the covariance.

$$Cov(\mathbf{x}, \mathbf{y}) = \frac{1}{N} \sum (x_n - \mu_{\mathbf{x}})(y_n - \mu_{\mathbf{y}}) \tag{25}$$

The covariance matrix can be represented as

$$\Sigma = \begin{bmatrix} Cov(\mathbf{x}_1, \mathbf{x}_1) & Cov(\mathbf{x}_2, \mathbf{x}_1) & ... & Cov(\mathbf{x}_n, \mathbf{x}_1) \\ Cov(\mathbf{x}_1, \mathbf{x}_2) & Cov(\mathbf{x}_2, \mathbf{x}_2) & ... & Cov(\mathbf{x}_n, \mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ Cov(\mathbf{x}_1, \mathbf{x}_n) & Cov(\mathbf{x}_2, \mathbf{x}_n) & ... & Cov(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix} \tag{26}$$

where $Cov(\mathbf{x}_1, \mathbf{x}_1) = Var(\mathbf{x}_1)$, which extends across the diagonal of the matrix.

To find the eigenvectors and eigenvalues of Σ, use

$$(\Sigma - \lambda I)\vec{e} = \vec{0} \tag{27}$$

Since $\vec{e} = \vec{0}$ would not yield a useful eigenvector, a solution must be found for $\det(\Sigma - \lambda I) = \vec{0}$.

Once we have calculated all of the eigenvectors of the covariance matrix, we can begin reducing to a lower dimension. The first step of this is to find the eigenvalues that best capture the variance of the data. To find these eigenvectors, we can use each eigenvector's corresponding eigenvalue. This can be done by creating a list of the eigenvalues, Λ, in descending order.

$$\Lambda = [\lambda_1, ..., \lambda_d] \tag{28}$$

where d is the number of dimensions or features, and $\lambda_1 \geq \ldots \geq \lambda_{d-1} \geq \lambda_d$

Then, we can find the percentage each eigenvalue represents of the total, or the explained variance using

$$\text{Explained Variance} = 100 \left(\frac{1}{\sum \lambda_n} \lambda_n \right) \tag{29}$$

where W_n represents an individual eigenvalue.

The eigenvectors of the covariance matrix are normally called the principal components. The top few principal components can be used to reduce the dimensions of the dataset.

9.1.2 Determinant of Covariance Matrix

The covariance is the relationship of two random variables. A positive covariance means that the variables tend to vary in the same direction, while a negative covariance would vary in opposite directions. Very similar to the correlation of two variables, except the covariances are not bounded between -1 and 1. Covariances are often organized into a matrix in which the determinant can be calculated. The determinant is a scalar value that indicative of how a matrix behaves and its properties.

Numbers within the matrix are denoted by $|C_{ij}|$ with the first as the row, second the column, e.g., $|C_{2,2}|$ would be a covariance of size $[2 \times 2]$.

The determinant of a 2x2 matrix is given by

$$\det(C) = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = (a * d) - (b * c)$$

However, for matrices of n x n size, can be determined with the Leibniz formula as follows:

$$\det(A) = \sum_{\sigma \in \mathbf{S}_n} (sgn(\sigma) \prod_{i=1}^{n} a_{i\sigma_i})$$

$\prod$ is the product of sequence of products within the indexes. Sgn or sign (σ) is the number of permutations required for the matrix to return to the column order from least to greatest. If the number of permutation is even, then the sign of the product is positive. If the number of permutations is odd, then the sign of the product is negative.

9.1.3 Eigenvectors for the Iris Dataset

The Iris dataset is a dataset with 150 observations and three different species. Each observation includes the factors sepal length $(x_{1,1}, x_{2,1}, \ldots, x_{10,1})$, sepal width $(x_{1,2}, x_{2,2}, \ldots, x_{10,2})$, petal length $(x_{1,3}, x_{2,3}, \ldots, x_{10,3})$, and petal width $(x_{1,4}, x_{2,4}, \ldots, x_{10,4})$. For now, we will calculate the eigenvectors and eigenvalues of the first 10 observations of the setosa species.

$$\begin{aligned}
\mathbf{x}_1 &= [5.1, 3.5, 1.4, 0.2] \\
\mathbf{x}_2 &= [4.9, 3.0, 1.4, 0.2] \\
\mathbf{x}_3 &= [4.7, 3.2, 1.3, 0.2] \\
\mathbf{x}_4 &= [4.6, 3.1, 1.5, 0.2] \\
\mathbf{x}_5 &= [5.0, 3.6, 1.4, 0.2] \\
\mathbf{x}_6 &= [5.4, 3.9, 1.7, 0.4] \\
\mathbf{x}_7 &= [4.6, 3.4, 1.4, 0.3] \\
\mathbf{x}_8 &= [5.0, 3.4, 1.5, 0.2] \\
\mathbf{x}_9 &= [4.4, 2.9, 1.4, 0.2] \\
\mathbf{x}_{10} &= [4.9, 3.1, 1.5, 0.1]
\end{aligned}$$

where each $\mathbf{x}_n$ is an individual observation and the features are represented as follows:

$$\begin{aligned}
F_{sl} &= [5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9] \\
F_{sw} &= [3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1] \\
F_{pl} &= [1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5] \\
F_{pw} &= [0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1]
\end{aligned}$$

First, calculate the mean by feature which results in $\boldsymbol{\mu} = [\mu_{sl}, \mu_{sw}, \mu_{pl}, \mu_{pw}]$ where $\mu_{sl} = 4.86$, $\mu_{sw} = 3.31$, $\mu_{pl} = 1.45$, and $\mu_{pw} = 0.22$.

Then, calculate the covariance matrix

$$\Sigma = \begin{bmatrix} 0.0764 & 0.0634 & 0.0170 & 0.0128 \\ 0.0634 & 0.0849 & 0.0155 & 0.0158 \\ 0.0170 & 0.0155 & 0.0105 & 0.0020 \\ 0.0128 & 0.0158 & 0.0020 & 0.0056 \end{bmatrix}$$

The covariance matrix can then be used to calculate the eigenvectors and eigenvalues

$$0.1509 \begin{bmatrix} -0.6679 \\ -0.7131 \\ -0.1616 \\ -0.1386 \end{bmatrix}, 0.0176 \begin{bmatrix} -0.7048 \\ 0.6782 \\ -0.1756 \\ 0.1118 \end{bmatrix}, 0.0067 \begin{bmatrix} 0.2272 \\ -0.0484 \\ -0.9400 \\ 0.2500 \end{bmatrix}, 0.0022 \begin{bmatrix} -0.0742 \\ -0.1709 \\ 0.2440 \\ 0.9517 \end{bmatrix}$$

In this case (because of the use of the NumPy module) the eigenvalues, Λ, are already sorted in descending order.

$$\Lambda = [0.1509, 0.0176, 0.0067, 0.0022]$$

With these eigenvalues, the explained variance can be calculated

$$\textbf{Explained Variance} = [85.0416, 9.9217, 3.7522, 1.2845]$$

Given the first 10 values of each of the setosa features in the iris dataset, we can capture 85.0416% of the data using one principal component.

Python Code

```python
import numpy as np
from numpy import linalg as la

class Eigen:
    def makeList(data, row):
        npList = data[row, :]
        return npList.tolist()

    def meanCalc(data, row):
        """ Find the mean of a dataset

        Keyword arguments:
        data -- the 2d numpy array
        row -- feature of data to calculate mean on

        """
        dataList = Eigen.makeList(data, row)

        sum = 0
        for item in dataList:
            sum += item
        n = len(dataList)
        mean = sum/n
        return mean

    def covarianceCalc(data, row1, row2):
        """ Find the covariance matrix of a 2d numpy array

        Keyword arguments:
        data -- the 2d numpy array
        row1, row2 -- the features of the dataset to calculate covariance on

        """

        dataList1 = Eigen.makeList(data, row1)
        dataList2 = Eigen.makeList(data, row2)

        mean1 = Eigen.meanCalc(data, row1)
        mean2 = Eigen.meanCalc(data, row2)

        sum = 0
        for i, item in enumerate(dataList1):
            zeroMean1 = dataList1[i] - mean1
            zeroMean2 = dataList2[i] - mean2

            sum += zeroMean1 * zeroMean2

        # population covariance, N, not N-1
        n = len(dataList1)

        covariance = sum/n

        return covariance

    def covarianceMatrix(data):
        """ Find the covariance matrix of a 2d numpy array

        Keyword arguments:
        data -- the 2d numpy array

        """

        numRows = np.shape(data)[0] #

        covarianceMatrix = np.empty(shape = [numRows, numRows])

        for i, row in enumerate(covarianceMatrix):
            for j, col in enumerate(row):
                covarianceMatrix[i][j] = Eigen.covarianceCalc(data, i, j)

        return covarianceMatrix

    def explainedVariance(eigenvalues):
        """ Find the Explained Variance of a list of eigenvalues

        Keyword arguments:
        eigenvalues -- the values for the Explained Variance to be calculated on

        """

        sum = 0
        for value in eigenvalues:
            sum += value

        explainedVariance = []

        for value in eigenvalues:
            explainedVariance.append((value/sum) * 100)

        return np.array(explainedVariance)


```

```python
    def combined(data):
        """ Find the Explained Variance of a dataset

        Keyword arguments:
        data -- the data for the Explained Variance can be calculated on
        """

        covarianceMatrix = Eigen.covarianceMatrix(data)
        w, v = la.eig(covarianceMatrix)
        explainedVariance = Eigen.explainedVariance(w)
        return explainedVariance
```

- show how to zero mean the data
- show how to take the covariance of the zero mean-ed data
- show how to calculate the Eigenvectors and Eigenvalues
- show how to sort the Eigenvalues and Eigenvectors
  - After sorting the Eigenvalues and Eignevectors
  - Sum the Eigenvalues $\Lambda_{sum} = \sum \Lambda$
  - Now take Λ and divide each of the values by Λ_{sum}
  - $\Lambda_{total} = \frac{\Lambda}{\Lambda_{sum}}$
  - $\sum \Lambda_{total} \times 100 = 100$
  - This will then lead to the variance explained based on percentage, for example, $\Lambda_{total} = [0.750.150.0750.025]$
  - So 90% of the variance are the top two Eigenvalues
- Show how to calculate the variance explained from based on the Eigenvalues

This now leads to the next section!!!

9.1.4 Number of Components to Retain (Dillon and Goldstein, 1984)

Principal component analysis is frequently employed for the purpose of generating a reduces set of variates that account for most of the variability in the original data, and that can be used in more substantial subsequent analysis. We must therefore decide just how many components to retain. Unfortunately there is not universally accepted method for doing so. The decision is largely judgmental and a matter of taste.

A number of procedures for determining how many components to retain have been suggested. These "rules" range from methods that evoke formal significance tests to less formal approaches involving heuristic graphical arguments. The remainder of this section discusses several of the more commonly used procedures.

Variance - Covariance Input A reasonable approach for determining the number of components to retain when factoring a variance-covariance matrix relies on the sampling distribution results. For example, we might suggest that only those components whose associated eigenvalues are statistically different from zero be retained.

The reader should note, however, that with even moderate sample sizes many of the components will typically be statistically significant. Yet, from a practical viewpoint, some of these significant components account for only a vary small proportion of the total variance. Hence, for reasons of parsimony and practical significance, the statistical criteria should be interpreted loosely with fewer components retained than are statistically significant. In this regard,, some of the graphical procedures soon to be discussed may prove useful.

An alternative and somewhat more ad hoc approach is the percentage of variance criterion. With this approach, the cumulative percentage of eh variance extracted by successive components is the criterion. Note, the cumulative proportion of total variance extracted by a set of components sis given by

$$\frac{\sum_{j=}^{m} l_{(j)}}{\sum_{j=}^{p} l_{(j)}} \tag{30}$$

where $m < p$. The stopping rule is subjective, since the number of components retained is based on some arbitrary determined criterion for the amount of variation accounted for.

Correlation Input When factoring a correlation matrix, statistical testing procedures no longer apply, and the retained variance criterion lacks clear meaning. For these reasons, various graphical heuristics have been suggested as a rule of thumb.

Perhaps the most frequent used extraction approach is the "root greater than one" criterion. Originally suggested by Kaiser (1958) [25], this criterion retains those components whose eigenvalues are greater than one. The dimensionality assessment is accomplished by extracting the principal components from the correlation matrix and retaining only the factors described in Kaiser's criterion (eigenvalues: $\lambda \geq 1$). Principal component analysis partitions the data by variance using linear combination of 'original' factors. The rationale for this criterion is that any component should account for more "variance" than any single variable in the standardized test score space.

Another approach, proposed by Cattell (1966) [26], is called the scree test. With this approach, named after the rubble at the bottom of a cliff, the eigenvalues of each component are plotted in successive order of their extraction, and then an elbow in the curve is identified by applying, say, a straightedge to the bottom portion of the eigenvalues to see where they form an approximate straight line. The number of components retained is given by the point at which the components curve above the straight line formed by the smaller eigenvalues. Cattell and Jaspers (1967) [27] suggested that the number of factors be taken as the number immediately before the straight line begins. The rationale for the scree test is simple: since the principal component solution extracts components in successive order of magnitude, the substantive factors appear first, followed by the numerous trivial components which account for only a small proportion of the total variance.

Though this approach is relatively simple, complications can surface. First, there may be no obvious break, in which case the test is inclusive. Second, There may be several breaks. This is particularly troublesome when, say, two breaks occur among the first half of the eigenvalues, since it will be difficult to decide which of the breaks reflects the correct number of components.

Horn (1965) [28] has suggested yet another approach for determining the number of components to retain which removes much of the ambiguity associated with the scree test. As in the scree test, the data are factored asn te resulting component eigenvalues are plotted in sucessinve order of their magnitude. Next, K setts of $n \times p$ normally and independently distributed (NID) random variates are sampled from a population whose correlation structure is known to be characterized by an identity matrix. Each $n \times p$ data matrix is factored. The average eigenvalue for each extracted component over the k sets of randomly generated data s then computed and plotted in the same graph with the real data. Because of samling variation, several of the extracted eigenvvalues may exceed unity. However, the average curve of eighenvalues can be expressed to cross the ordinate scale att the value 1.0 for component number $p/2$. The proincipal component solution for the simulatioed data represents the case in which the eigenvalues under the null hypothesis are unity. Consequently, Horn's criterion is to retain components n the basis of where the reference curve crosses the curve induced from the actual data.
Great Online Resource: http://sebastianraschka.com/Articles/2014_pca_step_by_step.html
https://blogs.sas.com/content/iml/2017/08/02/retain-principal-components.html

10 References

References

[1] G. M. Dillon, W. R., *Multivariate Analysis Method and Applications.* New York, NY: John Wiley Sons, Inc, 1984.

[2] S. Theodoridis and K. Koutroumbas, *Pattern Recognition.* Academic Press, 3rd ed., 2006.

[3] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, 2003.

[4] C. J. Richards, N. Valliani, B. A. Johnson, N. K. K. Wong, A. Pennati, A. K. Saeed, and B. M. Rodriguez, "Multimodal data fusion using signal/image processing methods for multi-class machine learning," in *Signal Processing, Sensor/Information Fusion, and Target Recognition XXXII* (I. Kadar, E. P. Blasch, and L. L. Grewe, eds.), vol. 12547, p. 125470N, International Society for Optics and Photonics, SPIE, 2023.

[5] B. M. Rodriguez, *Multi-Class Classification for Identifying JPEG Steganography Embedding Methods.* PhD thesis, Air Force Institute of Technology, 2008.

[6] T. L. Vic Barnett, *Outliers in Statistical Data.* Academic Press, 3rd ed., 1994.

[7] S. S. Wilks, *Mathematical Statistics.* New York London: John Wiley Sons, Inc., 1963.

[8] P. C. Mahalanobis, "On the generalised distance in statistics," vol. 2, pp. 49–55, 1936.

[9] A. C. Rencher, *Methods of Multivariate Analysis.* Wiley-Interscience, 2nd ed., 2002.

[10] K. Fukunaga, *Introduction to Statistical Pattern Recognition.* Academic Press, 2nd ed., 1990.

[11] "On a measure of divergence between two statistical populations defined by their probability distribution," 1943.

[12] R. Fisher, "The use of multiple measurements in taxonomic problems," *Proceedings of Annals of Eugenics*, no. 7, pp. 179–188, 1936.

[13] H. van der Heijden, "User acceptance of hedonic information systems," *MIS Quarterly*, 2004.

[14] C. Bishop, *Neural Networks for Pattern Recognition.* Oxford University Press, 1st ed., 1996.

[15] C. Bishop, *Pattern Recognition and Machine Learning.* Springer, 2006.

[16] D. W. Ruck, S. K. Rogers, M. Kabrinsky, and M. E. Oxley, "The multilayer perceptron as an approximation to a bayes optimal discriminant function," *IEEE Transactions on Neural Networks*, 1990.

[17] "Determining input features for multilayer perceptrons," *Neurocomputing*, 1995.

[18] I. Guyon, C. Aliferis, and A. Elisseeff, *Computational Methods of Feature Selection.* Chapman and Hall, 1st ed., 2007.

[19] J. Weston, A. Elisseeff, B. Schölkopf, and M. Tipping, "Use of the zero-norm with linear models and kernel methods," *Journal of Machine Learning Research 3*, 2003.

[20] A. Rakotomamonjy, "Variable selection using svm-based criteria," *Journal of Machine Learning Research 3*, 2003.

[21] N. Louw and S. Steel, "Variable selection in kernel fisher discriminant analysis by means of recursive feature elimination," *Computational Statistics Data Analysis*, 2006.

[22] H. Hotelling, "Analysis of a complex of statistical variables into principal components," *Journal of Educational Psychology*, vol. 24, pp. 417–441, 1933.

[23] B. Schölkopf, A. Smola, and K.-R. Muüller, "Nonlinear component analysis as a kernel eigenvalue problem," 1996.

[24] H. F. Kaiser, "The application of electronic computers to factor analysis," *Educational and Psychological Measurement*, vol. 20, no. 1, pp. 141–151, 1960.

[25] H. F. Kaiser, "The varimax criterion for analytic rotation in factor analysis," *Psychometrika*, 1958.

[26] R. B. Cattell, "The scree test for the number of factors," *Multivariate Behavioral Research*, 1966.

[27] R. B. Cattell and J. Jaspers, "A general plasmode (no. 30-10-5-2) for factor analytic exercises and research.," *Multivariate Behavioral Research Monographs*, 1967.

[28] J. L. Horn, "A rationale and test for the number of factors in factor analysis.," *Psychometrika*, 1965.

[29] B. J. S. C. Wayne Brown, *Graphics File Formats: Reference and Guide*. Manning Publications, 1995.

[30] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, 3rd ed., 2009.

[31] J. d'Alambert, *Researchers sur diferentes points importants du systeme du monde*. 1754.

[32] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. Wiley-Interscience, 2nd ed., 2000.

[33] P. Duhamel and H. Hollmann, "Existence of a 2n fft algorithm with a number of multiplications lower than 2n+1," 1984.

[34] Duin, R. P.W., Tax, David, Pekalska, and Elzbieta, "Prtools." `https://cmp.felk.cvut.cz/cmp/software/stprtool/index.html`.

[35] "Home of the jpeg committee," 2004.

[36] L. Euler, "Nova acta acad. sci. petrop," 1960.

[37] V. Franc and V. Hlavac, "Statistical pattern recognition toolbox." `https://cmp.felk.cvut.cz/cmp/software/stprtool/index.html`.

[38] H. H. Goldstine, *A History of Numerical Analysis from the 16th through the 19th Century*. Springer New York, 1977.

[39] "Machine learning at waikato university." `https://www.cs.waikato.ac.nz/~ml/index.html`.

[40] J. D. Murry and W. vanRyper, *Encyclopedia of Graphics File Formats: The Complete Reference on CD-ROM with Links to Internet Resources*. O'Reilly Media, 2nd ed., 1996.