Text

Description automatically generated with medium confidence

**Introduction**

**What is PCA:**

Principal Component Analysis (PCA) is an incredibly useful and widely used multivariate technique in Machine Learning. Moreover, such technique is also extremely helpful in the analysis of huge datasets, whilst effectively undertaking Dimensionality Reduction and Feature Selection. In continuation, PCA is used to ensure that data scientists would be able to load and use large datasets on less powerful machines, which could not support the size of the full dataset. Additionally, PCA also provides cleaner data visualisation through the envisioning of the key data features in the full dataset, which hold the largest degree of information. [1-2]

**History:**

PCA has a long and illustrious history that goes back more than a century. The algorithm was pioneered by Karl Pearson, who in 1901 launched this system with the aim of undertaking data analysis and dimensionality reduction. The current PCA's design was first provided by Harold Hotelling in the 1930s, but it wasn't until then that the method truly started to take shape.

Hotelling was instrumental in formulating the concept of variance maximization and the use of orthogonal projections to find the Principal Components. Further improvements to the PCA algorithm were developed in the 1960s, in part due to the emergence of Singular Value Decomposition (SVD), which offered an alternate method for calculating the eigenvalues and vectors necessary to do the PCA. The growing adoption of PCA at this time was largely compelled by the need for dimensionality reduction and the widespread growth of computers. Consequently, the method gained a lot of traction in the 1970s and later on when data scientists and researchers comprehended how effective it was at dealing with the enormous, complex datasets that were becoming more and more prevalent in industries such as banking, engineering, and medicine. [1-2]

**Motivation and Objectives:**

Nevertheless, such algorithm’s behaviour may not always be comprehensible, thus cementing the need for the creation of a visual tool, which could allow users to visualise the algorithm's stages and data transformations, whilst offering a better understanding on the modified data. Consequently, the programmed solution also effectively portrays the PCA process as a simple convenient story targeted towards students who have just completed a Linear Algebra or AI Numerical Methods course. Additionally, the created Jupyter Notebook which outlines the aforementioned process conveys to the students, the necessary information to be able understand such algorithm, whilst providing the students with essential tools to be able to experiment and expand their knowledge. In continuation, the notebook was also designed to be robust and responsive, allowing students to interact with the visual plots through the plot’s minimising and maximising tools. Accompanying the Jupyter Notebook, the developed solution comes equipped with various famous datasets utilised by the machine learning community such as the Iris dataset, with the aim to make the students familiar with such datasets. In addition, the datasets were chosen for their distinct properties, to allow students to evaluate different experiments and infer new knowledge.

**Summary of Results:**

The noteworthy key points attained from the creation of such artefact pertains to the following list:

1. Principle Component Analysis is designed to be utilised on linearly separable data i.e., the variables in the dataset need to be linked together through a linear relationship. Consequently, Kernel PCA can be used to resolve such limitation [3].
2. The PCA algorithm is intended to be used on continuous data values. The approach taken to cater for discrete value features in the dataset, included changing the discrete values to continuous values, through different encoding techniques. Moreover, such discrete columns could also be discarded before the calculation of the PCA, which would result in more accurate data projections, however incurring the loss of the discrete data columns.
3. There are different Libraries which may be used to implement PCA, for example the NumPy and the scikit-learn library. Furthermore, sometimes the NumPy library does not support the singular value decomposition of large datasets, and would sometimes crash, whilst, the scikit-learn library does not support such issue, as it utilises random singular value decomposition. In continuation, Randomized SVD can approximate the whole SVD with a substantially lower computation cost by randomly selecting a fraction of the matrix's rows or columns [4].
4. It is imperative that the data which is fed to the PCA algorithm is normalized, since if given unnormalized data (some data will have a high variance, and some will have a low variance), PCA will load on the high variance data [5]. Additionally, the created artefact utilises Z-Score Normalization.
5. Understanding the PCA algorithm mathematically can be quite a difficult task, as the process includes multiple mathematical calculations such as SVD or Covariance Matrix evaluation. Nevertheless, the erected artefact provides the need-to-know basis for such algorithm, in a well explained format.

**Background and Methodology**

Mathematically, PCA enables the conversion of linear continuous data into a new coordinate system, characterized by new axis (Principal Components) which are ordered in accordance with the features in the new coordinate system. Continuously, the best principal components can be plotted on different dimensional graphs, in order to provide a satisfactory visualisation of a large dataset. Unfortunately, such method may have some minimal data reduction, however visualising an n dimensional feature dataset on a 3D plot is quite a benefit. The PCA's main characteristics of decreasing the dimensionality of data, whilst retaining salient information, lead to it being the most effectively ranked data analysis and machine learning technique [1-2].

Figure 0 illustrates the various libraries which were used in the construction of the aforementioned notebook.

Graphical user interface

Description automatically generated with low confidence

Figure 0: Libraries utilised.

Continuously, the designed implementation provides depth to the above description of such algorithm, whilst partitioning the explanation in the following ordered sections:

[**1. Loading the Data**](http://localhost:8888/notebooks/IAPT/A%20Guide%20to%20Principal%20Component%20Analysis.ipynb#loadData)

A key step before the initiation of the PCA Algorithm involves the selection of a relevant dataset which will be analysed by such algorithm. Continuously, the designed implementation enables students interacting with the notebook the choice to select any of the default datasets and explore how the PCA algorithm will function on such datasets. Additionally, students are also given the option to load their preferred dataset. In addition, the default datasets chosen, were selected with the purpose of having different attributes. This was done, in order to allow students to carry out different experiments and be able to compare the results obtained through varying the datasets.

**The following are the default datasets (Obtained from [6-12]):**

1. **country\_wise\_latest.csv** - This dataset has a small Size, a large number of Features, and a few numbers of Discrete Columns.
2. **diabetes.csv** - This dataset has a small Size, a small number of Features, and no Discrete Columns.
3. **FIFA - 2014.csv** - This dataset has a small Size, a small number of Features, and one Discrete Column.
4. **IRIS.csv** - This dataset has a small Size, a large number of Features, and one Discrete Column.
5. **Salary\_Dataset\_with\_Extra\_Features.csv** - This dataset has a large Size, a small number of Features, and a reasonable number of Discrete Columns.
6. **spotify.csv**- This dataset has a large Size, a large number of Features, and a reasonable number of Discrete Columns.
7. **wine-quality-white-and-red.csv**- This dataset has a large Size, a large number of Features, and one Discrete Column.

The aforementioned functionality of loading the chosen dataset into a pandas data frame can be illustrated in Figure 1. Continuously in the provided code snippet, the user is presented with a Menu, and depending on the choice to load a preferred dataset or load a default dataset, the user will be given a relevant message to input the file path or name respectively. In case that the user selects an invalid option, the program will continue to loop, until the user has successfully inputted a valid input. In case that the user chooses to load a default dataset, the user is presented with the list of default csv files present in the Datasets folder. Consequently, retrieving such dataset names is executed dynamically through the os.listdir function, thus enabling adding another default dataset to the Datasets folder relatively simple. On the other hand, if the user decides to load a preferred dataset, a relevant check is issued to ascertain that the specified file exists. Finally, the specified file name is loaded through the pd.read\_csv function, and stored in a pandas data frame. In case the specified csv file has less than 3 columns, a Warning message is displayed to the user, as utilising such dataset will present Errors in the data visualisations later on. Additionally, calculating the PCA algorithm on a dataset which has less than three columns presents unpleasant results, notwithstanding applying the concept of dimensionality reduction on a dataset which already has reduced size, is quite redundant.

Text

Description automatically generated

Figure 1: Code required for Loading the chosen dataset into a pandas data frame.

[**2. Dataset Feature Selection**](http://localhost:8888/notebooks/IAPT/A%20Guide%20to%20Principal%20Component%20Analysis.ipynb#featureSelection)

Another key step when performing a data analysis or a machine learning study, pertains to observing the type and number of different **Genes/Features**, which the dataset has. Furthermore, this step is relatively important, as sometimes processing a huge number of features in the dataset may cause memory allocation issues or prolong the processing time of algorithms. Consequently, in this stage students are given the option to choose which features to retain from the dataset, through the form of a user input menu. Additionally, in case less than three columns are chosen, the first three columns will be added to the filtered dataset. This was applied as a fail-safe measure, to ensure that the filtered dataset, would have enough features for visualisation in the upcoming sections.

The code snippet illustrated in Figure 2 portrays the aforementioned functionality, whereby the program first creates a new copy of original data frame, and then proceeds to loop through all of the new data frame’s columns. For each column iteration, the program probes the user with the choice of keeping the current column or discarding it. In case that the user doesn’t wish to keep the current column, the pd.drop method is being utilised to remove such feature from the new data frame. At the end of such snippet, a while loop is being used to ensure that the new data frame has at least three columns. In continuation, for every iteration in the while loop, the pd.insert method is being used to add a column from the old data frame.

Text

Description automatically generated with low confidence

Figure 2: Code required for Feature Selection.

[**3. Dealing with Discrete Data**](http://localhost:8888/notebooks/IAPT/A%20Guide%20to%20Principal%20Component%20Analysis.ipynb#discreteData)

As previously mentioned, PCA is designed to be utilised on continuous data [13]. Thus, placing the need to transform discrete data into continuous data before using PCA on a dataset. This is accomplished, as discrete data lacks a continuous range of values and cannot be represented in the same way as continuous data for this cause.

There are various ways how discrete data can be Transformed/Encoded to continuous data, in order to be examined by the PCA. On the other hand, in case that students would prefer to discard such columns, and focus only on the continuous data, they could opt to remove such discrete columns from the data frame in the previous section. Additionally proper error checking was implemented, in case that the data frame does not include any discrete columns, ensuring the robustness of the developed solution.

**The following are different types of Encoders, which were implemented in the artefact:**

1. **One-Hot Encoding**
2. **Label Encoding**
3. **Ordinal Encoding (Similar to Label Encoding)**
4. **Count Encoding**
5. **Word Embeddings Model**

**One-Hot Encoding**  
One-hot Encoding is a data preparation technique used to transform discrete variables into a format that machine learning algorithms can examine. Consequently, this encoding algorithm works by creating a binary vector for each possible category in the data. Additionally, each binary vector would have a value of 1 or 0 to indicate the presence or absence of each category respectively. [14-16]

Additionally, such encoding technique is quite explosive, as the number of different Genes/Features obtained after applying One-Hot encoding on a single column, will greatly increase the number of columns depending on the number of distinct features in each column. For an algorithm which aims to reduce dimensionality, such approach to turn discrete data into continuous data is quite inefficient, notwithstanding the increase in memory and time complexity presented.

Continuously, one might think whether this binary vector can be transformed back to decimal. Note that such encoding algorithm exists and is known as Binary to Decimal Decoding. The aforementioned algorithm effectively transforms the binary vector back into a decimal value, thus reducing the size of the Genes/Features to their original number [17]. Essentially such encoding would take relatively more time whilst achieving the same results as Label Encoding or Ordinal Encoding.

In this section students are presented with an application of such encoding algorithm on the filtered data frame, and given a detailed explanation, why in the case of PCA such algorithm is inefficient. As illustrated in the code snippet of Figure 3, the program loops through all the filtered data frame columns and proceeds to check whether the current column has an Object type i.e., is a discrete column. In case the current column has such type, then the pd.get\_dummies function is applied on such column, and the result is stored inside a new data frame, whilst exiting the loop. Following this, the user is presented with a message displaying the difference in size between the encoded column and the original data frame.

Text

Description automatically generated

Figure 3: Code required for applying One-Hot Encoding on the first Discrete Column.

**Label Encoding**  
Label Encoding is another data preparation technique used to transform discrete variables into a format that machine learning algorithms can examine. Such encoder works by giving each distinct category a unique numeric value or code [14,15,18]. For instance, taking the list of categories ["hat","apple","cap"] will be encoded as [3,1,2] (as numeric values).

As can be depicted in code snippet of Figure 4, program first creates a new copy of filtered data frame, and then proceeds to loop through all of the new data frame’s columns. Continuously, for every column iteration, the program checks whether the current column has an Object type. In case the latter holds, the pd.factorise function is being used to apply Label Encoding on such column. Furthermore, for this encoding implementation the sort flag was set to True in the pd.factorise function, as to enable the transformed data to be assigned numerical values based on the sorted strings.

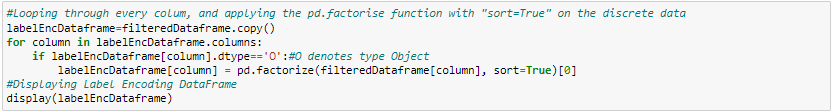


Figure 4: Code required for applying Label Encoding on the filtered data frame.

**Ordinal Encoding**  
Ordinal Encoding is another data preparation technique used to transform discrete variables into a format that machine learning algorithms can examine, which works similarly to Label Encoding. Such encoder works by giving each distinct category a unique numeric value or code, based on the order which the category appeared first [14,15,19]. For instance, taking the list of categories ["hat","apple","cap"] will be encoded as [1,2,3] (as numeric values, and encoded in the order which they appeared).

As can be seen in the code snippet of Figure 5, Ordinal Encoding is being implemented similarly to the Label Encoding implementation depicted in Figure 4. Notably the difference between both figures pertains that the pd.factorise function used to apply Ordinal Encoding, has the sort flag applied to False. Consequently, it was implemented in such a way as to enable the transformed data to be assigned numerical values based on the order in which the word appeared. Through the implementation of Label Encoding and Ordinal Encoding, which utilise the same function with some minor tweaks, students are given further opportunities to test out different encoding techniques, on the same dataset.

Text

Description automatically generated with medium confidence

Figure 5: Code required for applying Ordinal Encoding on the filtered data frame.

**Count Encoding**  
Count Encoding is another data preparation technique used to transform discrete variables into a format that machine learning algorithms can examine. Such encoder works by encoding each distinct category, with the number of times such category appeared [14-16]. For instance, if the category "hat" appeared 5 times, then "hat" will be encoded by the number 5.

The code snippet seen in Figure 6, depicts the implementation of the aforementioned Count Encoding technique, which utilises the same logic of realising discrete columns as the previously mentioned encoding techniques. Furthermore, the value\_counts function, is being used to count the number of times, each discrete value appears in the current discrete column. Afterwards, the pd.map function is being used to map the acquired array of discrete value frequencies to the current column’s discrete values. Ultimately transforming the discrete values to the number of times which they have appeared in the discrete column.

Text

Description automatically generated

Figure 6: Code required for applying Count Encoding on the filtered data frame.

**Word Embeddings Model**  
A Word Embeddings Model is a type of natural language processing (NLP) model which depict words as numerical vectors in a high-dimensional space. This model works by first training a neural network on a large corpus of text data, in order to represent words as dense, low-dimensional vectors. Each component of the word vector represents a specific aspect or characteristic of the word, such as its semantic meaning, part of speech, or syntactic context [20]. Moreover, the developed artefact focuses on the use of Word2vec, which is a type of Word Embeddings Model [20].

The code snippet seen in Figure 7, illustrates the implementation of the Word Embeddings Model Encoding technique, which utilises the same logic of realising discrete columns as the previously mentioned encoding techniques. Moreover, such techniques were implemented through the use of the Word2vec and nltk word\_tokenize libraries. The program first proceeds to tokenise all the strings in the discrete column through the word\_tokenize library and save the tokens in an array. Next the tokens are fed to the Word2Vec model with the min\_count parameter set to 1, as to consider words with a count of 1. Consequently, the resulting word vector obtained in the previous step is being used in conjunction with the df.apply and np.mean methods to assign a relevant value to the discrete values in the current column. Moreover, in this step, resulting nan values from the word embeddings model classification, are transformed to 0 as to avoid errors later on. The aforementioned process may take some time to complete, and users are presented with a message indicating which current column is being processed as a form of indication to the algorithm’s progress.

Text

Description automatically generated

Figure 7: Code required for applying Word Embeddings Model on the filtered data frame.

Consequently, all of the aforementioned encoding techniques were designed, with the purpose to familiarise students with different encoding techniques. Moreover, the large stack of encoding techniques also provides students with the liberty to test out the different techniques on different datasets, as it could result that a particular technique would outperform the others for a specific dataset. Additionally, in this section, throughout the detailed explanation of all of the aforementioned encoding techniques, students are presented with a visual representation of the different transformed continuous data frames. Nevertheless, after observing the results obtained from the different techniques, students are given the choice on which encoding technique to utilise in the continuation of the following sections. The choice can be seen through the code snippet presented in Figure 8, whereby by default the Label Encoding technique is adopted.

Text

Description automatically generated with low confidence

Figure 8: Student Choice for the encoding technique to utilise.

[**4. Filtered Dataset Visualisations**](http://localhost:8888/notebooks/IAPT/A%20Guide%20to%20Principal%20Component%20Analysis.ipynb#initialDataSetVisualisations)

Visualisation is a useful tool, as it aids in the process of identifying visual patterns and characteristics in data. Subsequently, it is simpler for individuals to spot patterns and trends when data is represented visually than when it is presented in numerical or written form. Unfortunately, not all features can be visualised, as visualised data is limited to three dimensions, thus individuals need to choose which features to visualise, from a high-dimensional dataset with many features.

In this section students are presented with a list of features in the filtered dataset and are given the option to choose their preferred features for the three-dimensional variables. Consequently, the aforementioned variables will be used for different visualisations. Consequently, this is illustrated in Figure 9.

Text

Description automatically generated with medium confidence

Figure 9: Student Choice for the features to utilise for visualisation.

Moreover,

The plotly library was utilised in order to provide interactive plots and visualisations which allow students to zoom in or zoom out of the plots, enabling them to recognize certain data trends better. Additionally, the colour component utilised in the different visualisation, do not show any relationship between the variables, but are used as a marker in order to compare graph axis between 2D and 3D plots. Furthermore, there are some cases where some graphs would need to rerun the code cell in order to appear, this is due to the plotly limit of 10 graphs. Figure 10 depicts the code required to visualise a 3D Plot.

A picture containing text

Description automatically generated

Figure 10: Code required to visualise a 3D Plot.

[**5. Normalizing Data**](http://localhost:8888/notebooks/IAPT/A%20Guide%20to%20Principal%20Component%20Analysis.ipynb#normalizingData)

Normalization is the process of converting and scaling the numerical characteristics inside a dataset to enable the data to have a uniform range and distribution. Furthermore, normalization's primary objective is to guarantee that no feature dominates or has an excessively large impact on the model's performance. This process is key, in the calculation of the PCA, since if given unnormalized data, the PCA algorithm will load on the high variance data [5]. An example would be having two data variables, one having a value of 1 and the other having a value of 700, whereby the PCA algorithm will issue higher importance to the second value. Recalling the previous encoding techniques, encoding techniques such as Label Encoding or Ordinal Encoding will provide transformed variables with an uneven distribution, i.e., some values will have a large integer value and others will have a small integer value. Through normalization the aforementioned issues can be resolved, and the data values would all be converted into a uniform range.

The developed implementation focuses on utilising **Z-Score Normalization** or also known as **Standardization** [21], and such normalization technique can be constructed through the formula presented in Figure 11.

A picture containing text

Description automatically generated

Figure 11: Z-Score Normalization Formula.

### Additionally, the formula used to calculate Standard Deviation (σ) can be seen in Figure 12.

A picture containing diagram

Description automatically generated

Figure 12: Standard Deviation Formula.

Figure 13 illustrates the implementation of the Z-Score Normalization. Additionally such function utilises the mean() and std() function for the calculation of the mean and standard deviation for each column respectively. Moreover, the formula presented in Figure 11 is applied on the current data frame, whereby every element in the data frame is subtracted by the mean and then divided by the standard deviation. In the case that the standard deviation results to being 0, then the current value in the data frame is set to 0. In addition, through the use of df.replace function, nans are replaced to 0, to avoid the program from stopping abruptly later on, whilst retaining as much data as possible.

Text

Description automatically generated

Figure 13: Z-Score Normalization Implementation.

[**6. Normalized Dataset Visualisations**](http://localhost:8888/notebooks/IAPT/A%20Guide%20to%20Principal%20Component%20Analysis.ipynb#normalizedDataSetVisualisations)

The normalized data frame attained in the previous sections, is being plotted through the plotly library, whilst keeping the same dimensional variables chosen by the student as to allow comparisons between graphs.

Through comparisons with the original dataset visualisation and the normalized data visualisation, one might note that the normalized data is plotted on a smaller range of values, when compared to the original plots in the section above. Additionally, one can also notice how in the normalized plots, the data values are centred around zero. Additionally, one might also note that the colour of the points in the normalized dataset plot may have changed colour when compared to the original plot, as the data values are now centred around zero and the normalized plot is plotted on a smaller range of values.

[**A**](http://localhost:8888/notebooks/IAPT/A%20Guide%20to%20Principal%20Component%20Analysis.ipynb#pcaCovariance) **note on Subsetting the Data to be used to calculate PCA from first principles**

For the following two PCA approaches, a small subset of the entire dataset is taken, in case that the dataset has a larger size than a respective threshold. Additionally, the dataset is being reduced to a tenth of its size, whilst maintaining the number of columns, in order to aid the student to better understand the concept, and method of calculation, in case the dataset has a larger size than the respective threshold of 10000 rows. Figure 14 illustrates the aforementioned process.

A picture containing text

Description automatically generated

Figure 14: Taking a small subset of the dataset.

[**7. Understanding PCA - SVD Approach**](http://localhost:8888/notebooks/IAPT/A%20Guide%20to%20Principal%20Component%20Analysis.ipynb#pcaSVD)

Singular Value Decomposition (SVD) is a decomposition method which is utilised to factorise a matrix of **m x n** size into three components. The resultant components include **U** and 𝑉T, which are two orthonormal matrices, and Σ which is a diagonal matrix containing the singular values of the original matrix. Additionally, the size/magnitude of each singular value signifies the importance in explaining the data [22]. For example, a singular value of 10 will have a higher importance than a singular value of 5. Additionally, the calculation formula for SVD Matrix Decomposition, can be seen in Figure 15.

Text

Description automatically generated

Figure 15: SVD Calculation Formula.

The second step in the Calculation of PCA via SVD Approach, involves multiplying the U matrix by the Σ matrix. This is performed as the multiplication of 𝑈. Σ presents a matrix whose columns give the projections of the data points on each principal axis.

Continuously, the aforementioned algorithm was implemented through the use of the NumPy library, particularly the svd() function, which given a matrix as input, the function would output the resultant components. However, the Sigma matrix needed to be constructed, as such function only returns the list of eigenvalues in descending order, and not the required matrix. Moreover construction of such matrix was facilitated through the np.zeros() and np.diag() functions respectively. Figure 16 illustrates the aforesaid concept.

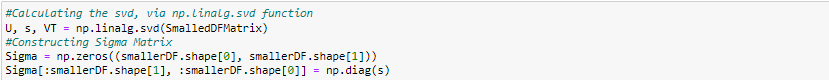


Figure 16: Code required for PCA Calculation via SVD Approach.

[**8. Understanding PCA - Covariance Matrix Approach**](http://localhost:8888/notebooks/IAPT/A%20Guide%20to%20Principal%20Component%20Analysis.ipynb#pcaCovariance)

A Covariance Matrix or also known as the **Covariance Variance Matrix** and is a **n x n** symmetric matrix which is used to show the covariance values between adjacent pairs of items in a dataset of n attributes. Additionally in this matrix, the diagonal elements represent the variance of each element. [23]

Consequently, the calculation formula for the Covariance Matrix, can be seen in Figure 17.

Table

Description automatically generated

Figure 17: Covariance Matrix Calculation.

In continuation, the principal components can be calculated through the multiplication of the normalised data frame with sorted eigenvectors of the Covariance Matrix. This is executed as the multiplication of the normalized data frame with the eigenvectors presents a matrix whose columns give the projections of the data points on each principal axis.

Figure 16 illustrates the aforementioned process, whereby the Covariance Matrix is calculated through the np.cov() function in the NumPy library, given the normalized data frame. Additionally, the eigenvectors and eigenvalues of the Covariance Matrix, are calculated through the NumPy.Linalg.eigh() function, and later sorted in descending order by eigenvalues. Finally, the principal components are calculated by applying matrix multiplication between the normalized data frame and the sorted eigenvectors, as depicted in Figure 16.

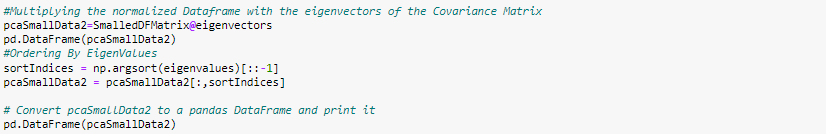
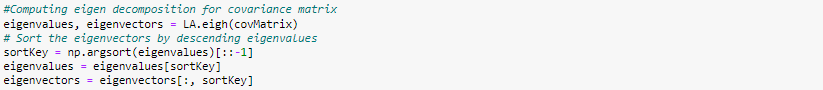


Figure 18: Code required for PCA Calculation via Covariance Matrix Approach.

[**A**](http://localhost:8888/notebooks/IAPT/A%20Guide%20to%20Principal%20Component%20Analysis.ipynb#pcaCovariance) **note on Calculating the Variance Ratio and Visualising ratio in Scree Plot**

The use of variance ratios in PCA is done, in order to calculate the percentage of the overall variance in the data that each principal component contributes to. Each principal component in the PCA algorithm captures a specific amount of data variation, thus one can determine the percentage of the overall variation that each component accounts for by computing the variance ratio. Additionally, through the variance ratio one is able to determine which principal components are crucial for explaining the variation in the data. [24]

**The Variance Ratio is calculated through the following Formula depicted in Figure 19 (obtained from [24-25]):**

Text

Description automatically generated with medium confidence

Figure 19: Variance Ratio Calculation.

The code snippet in Figure 20 depicts the calculation of the Variance Ratio, whereby firstly an array which holds the square of each principal component is created. Afterwards, the program loops through all the principal components, and for every iteration, the square for each principal component is updated. Next, the program proceeds to calculate the total sum of the calculated square values and storing the result in a variable. Finally, the variance ratio is calculated by dividing the array of square values corresponding to each principal component with the total square value. Additionally, the result is multiplied by 100, in order to obtain a percentage value.

Graphical user interface, text, application

Description automatically generated

Figure 20: Code required for Variance Ratio Calculation.

Moreover, through the calculated variance ratio can be displayed visually through the use of the Scree Plot. In continuation, through the use of such plot which displays the percentage of variation explained by each primary component, one is able to determine the number of components required to account for a specific percentage of the overall variance in the data [26]. Furthermore, example of such plot can be seen in Figure 21.

The following methods can be used to determine the optimal number of principal components to retain [26]:

1. **Elbow Method** - This method of selection adopts to retain all the principal components prior to the curve plateau in the Scree Plot. Moreover, this method works by pinpointing the point on the Scree Plot where the curve plateaus, and then selecting the number of components before this point as the ideal number of components to maintain.
2. **Kaiser Rule** - This method of selection selects to retain all the principal components with eigenvalues which have at least a value of 1.
3. **Proportion of Variance Plot** - This method of selection chooses to retain all the principal components which represent a percentage (%) amount of the variance.

Chart, histogram

Description automatically generated

Figure 21: Scree Plot example

[**9. Comparisons Between Approaches**](http://localhost:8888/notebooks/IAPT/A%20Guide%20to%20Principal%20Component%20Analysis.ipynb#approachCompare)

Although both approaches of computing the PCA algorithm essentially provide similar results, both approaches have their fair share of differences. For instance, in the SVD approach, one can compute the principal components directly by applying SVD decomposition on the original matrix. On the other hand, in the Covariance Matrix approach one needs to first compute the Covariance matrix, and then apply Eigen Decomposition in order to compute the principal components, making the process a lengthier one when compared to the SVD approach. Moreover, the Covariance approach also tends to be quite memory inefficient, due to the construction of the Covariance matrix, since the goal of the PCA is to reduce dimensionality, whilst in this approach one must first compute a larger matrix. Performance-wise, PCA with SVD outperforms PCA with covariance and is often quicker and more numerically stable. However, in some circumstances, such as when the data includes missing values or when the data is not centred, PCA with covariance may be chosen. [27]

In this section students are presented with the different Scree Plots and 2D Plots for the respective approaches next to each other, in order to grant the student, the opportunity to visually compare the results obtained together. Moreover, in some cases the visualisation of both approaches may have some discrepancies, as the graphs would be inverted copies of each other, due to the difference in the direction of the eigenvectors. Figures 22 and 23 illustrate an example of such discrepancy.

Chart, scatter chart

Description automatically generated

Figure 22: 2D Plot of PCA utilising SVD Approach.

Chart, scatter chart

Description automatically generated

Figure 23: 2D Plot of PCA utilising Covariance Matrix Approach.

[**10. Working out PCA on the Entire Dataset**](http://localhost:8888/notebooks/IAPT/A%20Guide%20to%20Principal%20Component%20Analysis.ipynb#pcaEntire)

The PCA algorithm implementations in the previous sections, were implemented for the sole purpose to educate Students on how the algorithm functions. Nevertheless, utilisation of such algorithm does not require the lengthy implementation in the previous sections, as one can easily adopt to use the PCA function in the scikit-learn library through: “from sklearn.decomposition import PCA” import. In continuation, a code snippet depicting the aforementioned concise implementation can be seen in Figure 24.

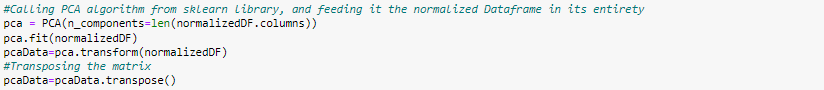


Figure 24: PCA algorithm through the scikit-learn library.

Additionally, through the creation of such notebook, it was discovered that sometimes the calculation of PCA via the NumPy library would crash the notebook when running on large datasets, whilst the PCA from the scikit-learn library does not. This is due, since in the scikit-learn library, if the input dataset has a size larger than 500 x 500, and the number of components to extract is less than 80% of the smallest dimension of the data, then a randomized SVD proposed by Halko [4] is utilised [28]. If not, the exact entire SVD is calculated and then could be truncated backwards [28]. Furthermore, utilisation of the NumPy library stores arrays in a contiguous block in memory, thus making the notebook crash in the case that the computer has insufficient memory [29].

In continuation, the Randomized SVD proposed by Halko, can approximate the whole SVD with a substantially lower computation cost by randomly selecting a fraction of the matrix's rows or columns [4]. Thus explaining, the mystery behind the enhanced efficiency in the scikit-learn approach, when compared to the NumPy approach.

[**11. PCA Visualisations**](http://localhost:8888/notebooks/IAPT/A%20Guide%20to%20Principal%20Component%20Analysis.ipynb#pcaVisualisations)

In this section students are presented with the relevant visualisations for the Scree Plot, 1D, 2D and 3D plots respectively. The required visualisations are plotted from the principal components obtained from applying the PCA algorithm of the scikit-learn library, on the whole dataset.

[**12. Conclusions and Limitations of PCA**](http://localhost:8888/notebooks/IAPT/A%20Guide%20to%20Principal%20Component%20Analysis.ipynb#conclusion)

**Evaluation**

**Conclusion**

**References**

[1] S. Mishra et al., "Multivariate Statistical Data Analysis-Principal Component Analysis," Int. J. Livest. Res., vol. 1, pp. 1-6, 2017. [Online]. Available: https://www.researchgate.net/publication/316652806\_Principal\_Component\_Analysis. [Accessed: 18-Apr-2023].

[2] D. Li and S. Liu, "4.2.3.1 Principal Component Analysis," in Water Quality Monitoring and Management: Basis, Technology and Case Studies, 1st ed., S. K. Gupta and R. Kumar, Eds. Amsterdam, Netherlands: Elsevier, 2019. [Online]. Available: https://www.sciencedirect.com/topics/agricultural-and-biological-sciences/principal-component-analysis. [Accessed: 18-Apr-2023].

[3] N. B. Subramanian, "Types of PCA", aiaspirant.com. [Online]. Available: <https://aiaspirant.com/types-of-pca/>. [Accessed: 18-Apr-2023].

[4] N. Halko, P. G. Martinsson, and J. A. Tropp, “Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions,” arXiv preprint arXiv:0909.4061, 2009. [Online]. Available: <https://arxiv.org/abs/0909.4061>. [Accessed: 18-Apr-2023].

[5] Stack Exchange. "Why do we need to normalize data before Principal Component Analysis (PCA)?", Cross Validated, May 26, 2014. [Online]. Available: <https://stats.stackexchange.com/questions/69157/why-do-we-need-to-normalize-data-before-principal-component-analysis-pca>. [Accessed: 18-Apr-2023].

[6] DEVAKUMAR K. P., "COVID-19 Dataset", Kaggle, 2020. [Online]. Available: <https://www.kaggle.com/datasets/imdevskp/corona-virus-report?select=country_wise_latest.csv>. [Accessed: 18-Apr-2023].

[7] UCI MACHINE LEARNING, "Pima Indians Diabetes Database", Kaggle, 2016. [Online]. Available: <https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database>. [Accessed: 18-Apr-2023].

[8] S. BANERJEE, "FIFA - Football World Cup Dataset", Kaggle, 2022. [Online]. Available: <https://www.kaggle.com/datasets/iamsouravbanerjee/fifa-football-world-cup-dataset?select=FIFA+-+2014.csv>. [Accessed: 18-Apr-2023].

[9] MATHNERD, "Iris Flower Dataset", Kaggle, 2018. [Online]. Available: <https://www.kaggle.com/datasets/arshid/iris-flower-dataset>. [Accessed: 18-Apr-2023].

[10] S. BANERJEE, "Software Industry Salary Dataset - 2022", Kaggle, 2022. [Online]. Available: <https://www.kaggle.com/datasets/iamsouravbanerjee/software-professional-salaries-2022>. [Accessed: 18-Apr-2023].

[11] R. Holbrook and A. Cook, "Principal Component Analysis, spotify.csv", Kaggle. [Online]. Available: <https://www.kaggle.com/code/ryanholbrook/principal-component-analysis/data?select=spotify.csv>. [Accessed: 18-Apr-2023].

[12] RUTHGN, "Wine Quality Data Set (Red & White Wine)", Kaggle, 2022. [Online]. Available: <https://www.kaggle.com/datasets/ruthgn/wine-quality-data-set-red-white-wine>. [Accessed: 18-Apr-2023].

[13] V. Karthik, "PCA for categorical features", Stack Overflow, Dec. 2016. [Online]. Available: <https://stackoverflow.com/questions/40795141/pca-for-categorical-features#:~:text=PCA%20is%20designed%20for%20continuous,yes%2C%20you%20can%20use%20PCA>. [Accessed: 18-Apr-2023].

[14] Datagy. "Pandas get\_dummies (One-Hot Encoding) Explained," Datagy.io, Feb. 2021. [Online]. Available: <https://datagy.io/pandas-get-dummies/>. [Accessed: 18-Apr-2023].

[15] DataCamp. "Dealing with Categorical Data". DataCamp, 2021. [Online]. Available: <https://www.datacamp.com/tutorial/categorical-data>. [Accessed: 18-Apr-2023].

[16] B. Roy, "All about Categorical Variable Encoding," Towards Data Science, Jul. 2, 2019. [Online]. Available: <https://towardsdatascience.com/all-about-categorical-variable-encoding-305f3361fd02>. [Accessed: 18-Apr-2023].

[17] T. Crosley, "What is the binary to decimal decoder?", Quora, May 8, 2018. [Online]. Available: <https://www.quora.com/What-is-the-binary-to-decimal-decoder>. [Accessed: 18-Apr-2023].

[18] Pandas. "pandas.factorize()". pandas 1.4.0 documentation, Jan. 07, 2022. [Online]. Available: <https://pandas.pydata.org/docs/reference/api/pandas.factorize.html>. [Accessed: 18-Apr-2023].

[19] J. Brownlee, "One-Hot Encoding for Categorical Data," Machine Learning Mastery, Aug. 17, 2020. [Online]. Available: <https://machinelearningmastery.com/one-hot-encoding-for-categorical-data/>. [Accessed: 18-Apr-2023].

[20] Vatsal, "Word2Vec Explained", Towards Data Science, Jul. 29, 2021. [Online]. Available: <https://towardsdatascience.com/word2vec-explained-49c52b4ccb71>. [Accessed: 18-Apr-2023].

[21] R. Sharma. "What is Normalization in Data Mining and How to Do It?", UpGrad, Sep. 22, 2022. [Online]. Available: <https://www.upgrad.com/blog/normalization-in-data-mining/#:~:text=Project%20Ideas%20%26%20Topics-,Z%2DScore%20Normalization,up%20to%20%2B3%20standard%20deviation>. [Accessed: 18-Apr-2023].

[22] M. E. Wall, A. Rechtsteiner, and L. M. Rocha, "Singular Value Decomposition and Principal Component Analysis," in Learning from Data: Concepts, Theory, and Methods, vol. 2, Springer, Boston, MA, 2007, pp. 151-176, doi: 10.1007/0-306-47815-3\_5. [Online]. Available: <https://www.researchgate.net/publication/2167923_Singular_Value_Decomposition_and_Principal_Component_Analysis>. [Accessed: 18-Apr-2023].

[23] CUEMATH, "Covariance Matrix", CUEMATH. [Online]. Available: <https://www.cuemath.com/algebra/covariance-matrix/>. [Accessed: 18-Apr-2023].

[24] I. T. Jolliffe and J. Cadima, "Principal component analysis: a review and recent developments," in The Data Deluge: Can Libraries Cope with E-Science? Proceedings of a Conference Held at the Royal Society, London, UK, 4-5 November 2004, vol. 463, Royal Society Publishing, 2016, pp. 21-36. doi: 10.1098/rsta.2015.0202.[Online]. Available: <https://royalsocietypublishing.org/doi/10.1098/rsta.2015.0202>. [Accessed: 18-Apr-2023].

[25] K. Guillaumier, "Linear Algebra in Data Science and PCA"

[26] S. Mangale, "Scree Plot," Medium, Aug. 28, 2020. [Online]. Available: <https://sanchitamangale12.medium.com/scree-plot-733ed72c8608>. [Accessed: 18-Apr-2023].

[27] M. E. Wall, A. Rechtsteiner, and L. M. Rocha, "Singular Value Decomposition and Principal Component Analysis," in Learning from Data: Concepts, Theory, and Methods, vol. 2, Springer, Boston, MA, 2007, pp. 151-176, doi: 10.1007/0-306-47815-3\_5. [Online]. Available: <https://www.researchgate.net/publication/2167923_Singular_Value_Decomposition_and_Principal_Component_Analysis>. [Accessed: 18-Apr-2023].

[28] Scikit-learn, “sklearn.decomposition.PCA", scikit-learn.org. [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html>. [Accessed: 18-Apr-2023].

[29] M. Kumar, "Memory error in NumPy SVD," in IEEE, 2014. [Online]. Available: <https://stackoverflow.com/questions/21180298/memory-error-in-numpy-svd>. [Accessed: 18-Apr-2023].