Optimizing Credit Card Fraud Detection Through Extensive Investigation of PCA.

Ravi Bhushan [1] Dr. Vineeta Khemchandani [2]

[1]School of Computer Science and Engineering.Galgotias University, Gr.Noida

[2]School of Computer Applications and Technology Galgotias University, Gr.Noida

Abstract

Principal component analysis (PCA) is a statistical technique that simplifies complex data sets by reducing the number of variables while retaining key information. PCA identifies new uncorrelated variables that capture the highest variance in the data. Principal Component Analysis (PCA) is one of the most widely used data analysis methods in machine learning and AI. This manuscript focuses on the mathematical foundation of classical PCA and its application to a small-sample-size scenario and a large dataset in a high-dimensional space scenario. Specifically, we explore a straightforward approach for approximating PCA in the latter scenario. We hope this manuscript will provide readers a solid foundation on PCA and approximate PCA. The research presented in this paper applies the principal component analysis (PCA) of the real-world credit card fraud detection dataset, gathered from the transactions of the European credit card users. The original dataset is highly imbalanced; to further analyse the performance of tuned machine learning models, we need to compose an imbalanced dataset into a balanced one. Therefore, we apply the principal component analysis (PCA) to the credit card dataset to determine the number of attributes needed for the implementation of machine learning models.

Keywords PCA , Eigen-decomposition, Approximate PCA, credit card fraud.

1 Introduction

Since the global pandemic of COVID-19 forced many economies to reevaluate the traditional onsite working environment paradigm, there has been a significant increase in e-commerce and online services based on credit card transactions. The bigger usage of credit cards was followed by an increased number of credit card frauds. This kind of criminal activity occurs when credit card authentication information is stolen with the malicious goal to buy merchandise or services without the owner's permission or to withdraw money from it. For this reason,

it is imperative to implement PCA to simplifies complex data sets by reducing the number of variables while retaining key information.
A common way to attempt to resolve this problem is to use dimensionality reduction techniques. One of the most popular techniques for this purpose are: Principal Components Analysis (PCA)). PCA has been used in face recognition [27], [21], [28], [23], [22], handprint recognition [24], human-made object recognition [25], industrial robotics [26], and mobile robotics [29].
Note that it is common to work in a very high-dimensional space in AI, machine learning, and data science. Some intuitions we have gained from our daily life experience in two or three dimensions may not be correct in this very high-dimensional space. Readers interested in properties of high-dimensional data may refer to [2]for more detail.
Principal Component Analysis (PCA) provides such a combination method. Indeed, PCA relies on *linear combination* of these features to construct *principal subspace* that is the main subspace on which most of the feature vectors lie. PCA uses *variance* as a measure of the information content of the subspace. The quality of the selected subspace can be measured by comparing the variance of data within it to the total variance of the whole dataset. These are two keys idea of PCA. In practice, however, one may run into practical problems such as hav- ing too few observations compared to the data dimension or having too much computational cost due to the high dimensionality of data and a large number of observations, etc. This paper describes some practical algorithms for handling these different scenarios. Some material pre- sented hereafter is extended from our previously published paper [13]. It should be noted that this tutorial focuses mainly on the algorithmic aspect of PCA. Readers interested in its applications are invited to consult the paper [7], [1] and references therein to see different use-cases of PCA on different domains including bioinformatics and quantum computing.
In the following, Sect. 2 first describes PCA in the usual setting where the number of avail- able data points is larger than the dimension of feature vectors. Then Sect. 3 describes PCA that is designed for small-sample-size case.. This section describes a variation of PCA designed to handle this case. Section 4 describes another variation of PCA designed for large-scale datasets in a very high-dimension space. Indeed, current AI and data science datasets are often large with very high-dimension feature vectors. This section explains how to mitigate the computational cost of PCA in this case.

2 Classical case: $n > d$

This section describes a classical scenario for applying PCA, namely when the number of available data points is larger than the dimension of feature vectors. In this section, we assume that the dimension of feature vectors is not very large, so all calculations can be done on a modest computer.

2.1 Notation

In the following, let $\mathbf{z} \in \mathbb{R}^d$ denotes a feature vector in d-dimensional space with

$$\mathbf{z} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$$

A feature vector contains values of features that are observed on actual data. We assume that there are no missing values. We further assume that these features are all numerical.

In theory, the features should be selected adequately by domain experts. Nonetheless, in practice, we often rely on low-level features, such as the intensity of each pixel in an image, that can be computed easily. High-level features, like loops and cusps in hand- written strokes, contain more information. However, their detection depends on image processing functions that must be put in place first. These functions may contain some heuristic or programming bias. As a result, for some poorly scanned images, the detec- tion of high-level features may not be done accurately. Low-level features, such as pixel intensities, are more robust because they can always be extracted from all input images. Even if each low-level feature may not convey a meaningful description by itself, com- bining lots of them with lots of data makes it possible to derive meaningful meaning using machine learning models. PCA is one method to achieve this goal.

2.2 Linear projection and dot product

The main idea of PCA is to use *variance* as a measure of information content and to identify *linear subspace* maximizing the variance. The linear subspace can be con- structed from multiple *unit vectors* that are orthogonal to each other. Each unit vec- tor represents a *projection axis*. Consider a projection axis $\mathbf{w} \in \mathbb{R}^d$, the projection of
$\mathbf{z} \in \mathbb{R}^d$ onto the axis is the point on this axis that is closest to . This projection can
be computed by $\|\mathbf{w}\| \cos \theta_{\mathbf{z},\mathbf{w}}$. Recall that the "*dot product*" or the "*scalar product*" or the "*inner product*" between any two vectors and denotes as $\langle \mathbf{u}, \mathbf{v} \rangle$ satisfies

$$\langle \mathbf{u}, \mathbf{v} \rangle = \mathbf{u}^T \mathbf{v} = \|\mathbf{u}\|\|\mathbf{v}\| \cos \theta_{\mathbf{u},\mathbf{v}}. \tag{1}$$

In the following, we will use ⟨**u**, **v**⟩ or $\mathbf{u}^T$ **v** interchangeably when talking about dot product. The projection of onto the unit vector axis is then given by a simple dot product $\mathbf{z}^T \mathbf{w} \in \mathbb{R}$. Note that the "projection" refers to the location *on the projection axis*. The same location can be described relative to the input space as well by multiplying it with the projection axis, i.e., ($\mathbf{z}^T$ **w**)**w**. This quantity is often referred to as the "reconstruction" of from its projection on the axis.

2.3 Variance on projection axis and covariance matrix

Given a set of centered real values $z_1, z_2, \dots, z_n$ $z_t \in \mathbb{R}$, the variance of this dataset is

$$\mathrm{Var} = \frac{1}{n} \sum_{t=1}^{n} z_t^2. \tag{2}$$

Given a set of feature vectors $\mathbf{z}_1, \dots, \mathbf{z}_n$, $\mathbf{z}_t \in \mathbb{R}^d$, and a projection axis (that is a unit vector, i.e. $\|\mathbf{w}\| = 1$). The projections on are $\mathbf{w}^T \mathbf{z}_1, \dots, \mathbf{w}^T \mathbf{z}_n$. Suppose that the feature vectors are centered around zero vectors. Thus, their linear projections will also be cen- tered appropriately. Note that each dot product yields a simple real value, i.e., $\mathbf{w}^T \mathbf{z}_t \in \mathbb{R}$. Hence, the Eq. (2) can be used to compute the variance:

$$\mathrm{Var}(\mathbf{w}) = \frac{1}{n} \sum_{t=1}^{n} (\mathbf{w}^T \mathbf{z}_t)^2 \tag{3}$$

(3)

$$= \frac{1}{n} \sum_{t=1}^{n} \mathbf{w}^T \mathbf{z}_t \mathbf{z}_t^T \mathbf{w} \tag{4}$$

$$= \mathbf{w}^T \left(\frac{1}{n} \sum_{t=1}^{n} \mathbf{z}_t \mathbf{z}_t^T \right) \mathbf{w}. \tag{5}$$

n

Covariance matrix

The last line gives us a shortcut to compute variance on any axis . Indeed, instead of projecting all data onto the axis and then computing the variance, we can *pre-compute* the covariance matrix **C** first, then given any projection axis   the variance of the projection on is given by $\mathbf{w}^T\mathbf{C}\mathbf{w}$.

2.4 PCA problem and Lagrangian

From the above calculation, we can formulate the main idea of PCA as the following con- strained optimization problem:

$$\max_{\mathbf{w}} \quad \mathbf{w}^T C\mathbf{w} \qquad (6)$$
$$\text{subject to } \mathbf{w}^T\mathbf{w} = 1.$$

For standard optimization problems, we may set the partial derivatives to zero and solve the resulting equations or follow the gradient. However, the above problem has an addi- tional constraint that must be considered. To this end, the standard approach is to use the *Lagrangian method* (see (Dimitri 1996) for more detail on the Lagrangian method). In short, to solve the following constrained optimization problem:

$$\max_{\mathbf{z}} \quad f(\mathbf{z})$$
$$\text{subject to } g(\mathbf{z})= 0, \qquad (7)$$

the following *Lagrangian function* is considered:

$$L(\mathbf{z},ß)= f(\mathbf{z})-ßg(\mathbf{z}), \qquad (8)$$

with *ß* the *Lagrangian coefficient* describing the important of the constraint. Then, given the solution ($\mathbf{z}^*$, $ß^*$) optimizing the function *L*, $\mathbf{z}^*$ will be the solution of the initial problem. For the PCA problem, its Lagrangian is given by

$$L(\mathbf{w},ß)= \mathbf{w}^T\mathbf{C}\mathbf{w} - ß\ \mathbf{w}^T\mathbf{w} - 1 \qquad (9)$$

Then, if we compute the partial derivative of *L* with respect to   and set it to zero, we obtain:

$$0 = \frac{6L}{} \qquad 6\mathbf{w}$$

(10)

$$=2\mathbf{Cw} - 2\beta\mathbf{w} \tag{11}$$

$$\mathbf{Cw} = \beta\mathbf{w}. \tag{12}$$

The above derivation uses *matrix calculus*. Readers unfamiliar with this calculation could consult reference (Petersen and Pedersen 2012) for more detail.

Eq. (10) means that the axis maximizing variance must be an *eigenvector* of **C**, and β is its corresponding *eigenvalue*. The eigenvectors and eigenvalues are, in fact, math- ematical objects that have been studied for a very long time. PCA represents another appli- cation of this well-founded subject. An example of a commonly used method to extract eigenvector and eigenvalue pair, or eigen-decomposition algorithm called the *Jacobi method* is shown in Section A.

2.5 Eigenvalue, variance and axis selection

From Sect. 2.4, we have seen that the projection axis maximizing variance must be an eigenvector of the covariance matrix **C**. However, as **C** is a $d \times d$ matrix, it has d eigenvec- tors; which one should be selected? To answer this question, we can observed that

$$\mathbf{Cv}_i = \beta_i \mathbf{v}_i \tag{13}$$

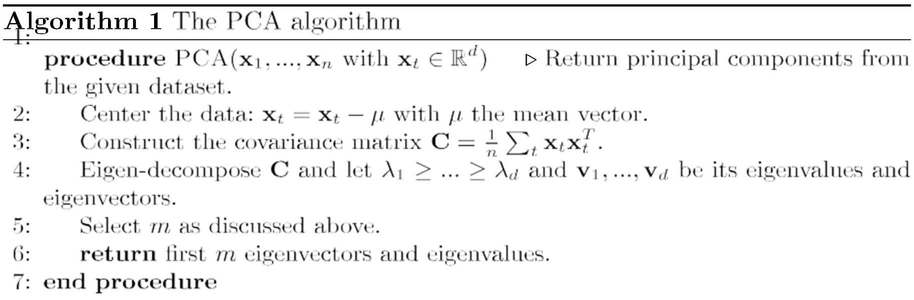
$$\mathbf{v}_i^T \mathbf{Cv}_i = \beta_i \mathbf{v}_i^T \mathbf{v}_i \tag{14}$$

$$\mathrm{Var}(\mathbf{v}_i) = \beta_i. \tag{15}$$

In other words, the eigenvalue corresponds to the variance of the data projected onto its corresponding eigenvector. Given eigenvalues $\beta_1 \geq \beta_2 \geq \ldots \geq \beta_d$ with $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_d$ the corresponding eigenvectors. From the discussion above, we know that if we select only one projection axis, then $\mathbf{v}_1$ should be the best choice since it corresponds to the largest eigenvalue, thus the largest variance of projected data.Note that from these eigenvalues we can easily compute the total variance, i.e. $\sum^d \beta . I$ .This total variance represents the entire information content of the dataset. By comparing β_1 to the total variance, we can judge if a single projection axis $\mathbf{v}_1$ is enough to retain the most information or not. In practice, a single projection axis is not enough. Fortunately, as the eigenvectors are already orthogonal, it is possible to select the next "largest" eigen- vectors to form the basis of the *principal subspace*. The next question is how many axes should be selected.

There are three approaches to this question:

1. User gives the desired number of axis *m*.
2. If we assume that *p* portion of the information content was, in fact, noise, then *m* should be the smallest number such that $\sum_{i=1}^{m} \beta_i \geq p \sum_{i=1}^{d} \beta_i$.
3. We can also assume that all axis with variance smaller than a pre-defined threshold are noise and should be discarded. Thus, we select all axis that $\beta_i \geq \varepsilon \beta_1$



Algorithm 1 The PCA algorithm

```
1: procedure PCA($\mathbf{x}_1, ..., \mathbf{x}_n$ with $\mathbf{x}_t \in \mathbb{R}^d$)    ▷ Return principal components from the given dataset.
2:     Center the data: $\mathbf{x}_t = \mathbf{x}_t - \mu$ with $\mu$ the mean vector.
3:     Construct the covariance matrix $\mathbf{C} = \frac{1}{n} \sum_t \mathbf{x}_t \mathbf{x}_t^T$.
4:     Eigen-decompose $\mathbf{C}$ and let $\lambda_1 \geq ... \geq \lambda_d$ and $\mathbf{v}_1, ..., \mathbf{v}_d$ be its eigenvalues and eigenvectors.
5:     Select $m$ as discussed above.
6:     return first $m$ eigenvectors and eigenvalues.
7: end procedure
```

The PCA algorithm is summarized in Algorithm 1. Given principal axis $\mathbf{v}_1$, ..., $\mathbf{v}_m$, each new data point can be represented within the principal subspace as:

$$\begin{bmatrix} \mathbf{z}^T \mathbf{v}_1 \\ \mathbf{z}^T \mathbf{v}_2 \\ \vdots \\ \mathbf{z}^T \mathbf{v}_m \end{bmatrix} \in \mathbb{R}^m, \quad (16)$$

and the reconstruction from its projection on the principal subspace is given by

$$\hat{\mathbf{z}} = \sum_{i=1}^{m} (\mathbf{z}^T \mathbf{v}_i) \mathbf{v}_i \quad (17)$$

It is pretty straightforward to prove that the selected principal subspace also minimizes the reconstruction error $\sum \|\mathbf{z} - \hat{\mathbf{z}}\|^2$ and that the reconstruction error is indeed the sum of all eigenvalues outside the principal subspace.

3 Small-sample-size case: $n \leq d$

This section describes the PCA method designed for the case where the number of data points (n) is smaller than the dimension of feature vectors (d). Even if the method described in this section originated from image application, it could also be used with general feature vectors when $n \leq d$. This scenario appears, for example, in bioinformat- ics [7]. In [7], the author studied 8,534 probes on the microar- rays with expression measurements extracted from 105 samples. In other words, the authors consider a dataset of 105 feature vectors in 8,534 dimensions. This dataset can be processed using the PCA algorithm described in this section.

In the following, Sects. 3.1 and 3.2 discuss the dot matrix, its relation to the covari- ance matrix, and the conversion of eigenvectors obtained from both matrices. The use of the dot matrix is the foundation of the small-sample-size PCA method described in Sect. 3.3.

3.1 Covariance matrix and Dot matrix

Given feature vectors $\mathbf{z}_1, \dots, \mathbf{z}_n$ with $\mathbf{z}_t \in \mathbb{R}^d$. We assume that these vectors are cen- tered around zero. We can construct a *data matrix* $\mathrm{X} = [\mathrm{z}_1, \dots, \mathrm{z}_n]$ of size $d \times n$. The column i of $\mathbf{X}$ corresponds to the feature vector $\mathbf{z}_i$. Using this data matrix, the covari-
ance matrix can be rewritten as

$$\mathbf{C} = \frac{1}{n}\sum_{t=1}^{n} \mathbf{z}_t\mathbf{z}_t^T = \frac{1}{n}\mathbf{X}\mathbf{X}^T. \tag{18}$$

We often use $\mathbf{XX}^T$ to denote the covariance matrix since maximizing $\mathbf{w}^T\mathbf{C}\mathbf{w}$ or $\mathbf{w}^T\,\mathbf{XX}^T\,\mathbf{w}$ will result in the same eigenvectors (if we constraint these eigenvectors to be unit vectors). The dot matrix is defined as $\mathbf{X}^T\mathbf{X}$. It is easy to see that the element i, j of this matrix is indeed the dot product between example i and j, i.e. $\mathbf{z}^T\mathbf{z}_j$.

3.2 Eigenvector conversions

Suppose that β and are eigenvalue and corresponding eigenvector of dot matrix, then we have

$$\mathbf{X}^T\mathbf{X}\mathbf{v} = \beta\mathbf{v} \tag{19}$$

$$(\mathbf{XX}^T)(\mathbf{Xv}) = \beta(\mathbf{Xv}). \tag{20}$$

Hence $\mathbf{Xv}$ will be an eigenvector of the covariance matrix under the same eigenvalue β. This relation allows us to convert the dot matrix's eigenvectors into the covariance matrix's eigenvectors. After the conversion, the newly obtained eigenvector should be normalized as usual. Note that we have

$$\mathbf{v}^{\mathbf{T}}\mathbf{X}^{\mathbf{T}}\mathbf{X}\mathbf{v} = \beta. \tag{21}$$

Hence the normalized version of the eigenvector is simply

$$(1/\sqrt{\beta})\mathbf{X}\mathbf{v}$$

On the other hand, it is also possible to convert eigenvectors of the covariance matrix into eigenvectors of the dot matrix. Indeed, if β and are eigenvalue and cor- responding eigenvector of dot matrix, then we have

$$\mathbf{X}\mathbf{X}^{T}\mathbf{u} = \beta\mathbf{u} \tag{22}$$

$$(\mathbf{X}^{T}\mathbf{X})(\mathbf{X}^{T}\mathbf{u}) = \beta(\mathbf{X}^{T}\mathbf{u}). \tag{23}$$

and $\mathbf{v}_1, ..., \mathbf{v}_d$ be its eigen

Hence $\mathbf{X}^{T}\mathbf{u}$ will be an eigenvector of the dot matrix under the same eigenvalue β. To nor- malize this eigenvector, we have

$$\mathbf{u}^{T}\mathbf{X}\mathbf{X}^{T}\mathbf{u} = \beta. \tag{24}$$

Hence the normalized version of the eigenvector is

$$1/\sqrt{\beta})\mathbf{X}^{T}\mathbf{u}$$

Algorithm 2 The small-sample-size PCA algorithm

```
1: procedure SSS-PCA($\mathbf{x}_1, ..., \mathbf{x}_n$ with $\mathbf{x}_t \in \mathbb{R}^d$ with $d > n$)    ▷ Return principal components from the given dataset.
2:     Center the data: $\mathbf{x}_t = \mathbf{x}_t - \mu$ with $\mu$ the mean vector.
3:     Construct the dot matrix $\mathbf{X}^T\mathbf{X}$.
4:     Eigen-decompose dot matrix and let $\lambda_1$
       values and eigenvectors.
5:     Perform conversion: we obtain $(1/\sqrt{\lambda_1})\mathbf{X}\mathbf{v}_1, ..., (1/\sqrt{\lambda_d})\mathbf{X}\mathbf{v}_d$ eigenvectors of the covariance matrix.
6:     Select $m$ as discussed above.
7:     return first $m$ eigenvectors and eigenvalues.
8: end procedure
```

3.3 PCA from dot matrix

From the above discussion, the small-sample-size PCA can be done by eigen-decompose the dot matrix first, then converting the result into eigenvectors of the covariance matrix. The whole procedure is summarized in Algorithm 2. From the computational point of view, the rule of thumb is to eigen-decompose a smaller matrix. If $d \leq n$, then we should eigen-decompose the dot matrix, if $d > n$ we eigen-decompose the covariance matrix.

4 Large-scale dataset in high-dimensional space: large n and large d

As time goes by, the size of dataset increases from a few hundred data to more than ten thousand data. Consequently, both covariance matrix and dot matrix will be huge and become difficult to eigen-decompose. This section describes how to deal with this problem, starting from a simple heuristic designed for data (Sect. 4.1), then we discuss why it works (Sect. 4.2) before giving a general method to handle this case in Sect. 4.4.

4.1 Approximate PCA

In summary, the PCA for large-scale data in high dimensional space can be done by adapt- ing the Algorithm 3 using dot product preserving transformation instead of simple down- sizing. This approximate PCA algorithm is summarized in Algorithm 4

Algorithm 4 The approximate PCA algorithm

```
1:  procedure APPROX-PCA($\mathbf{x}_1, ..., \mathbf{x}_n$ with $\mathbf{x}_t \in \mathbb{R}^d$ with large $d$ and large $n$)   ▷
    Return approximate principal components from the given dataset.
2:      Center the data
3:      $\tilde{\mathbf{x}} = f(\mathbf{x}) \in \mathbb{R}^q$ with $f$ dot product preserving function, e.g. image resizing,
    random projection, feature hashing.
        $q \ll n$ and $\tilde{\mathbf{X}} = [\tilde{\mathbf{x}}_1, ..., \tilde{\mathbf{x}}_n]$
4:      Eigenstructure of new covariance matrix $\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T$: $\lambda_1 \geq ... \geq \lambda_q$ and $\mathbf{u}_1, ..., \mathbf{u}_q, \mathbf{u}_i \in$
    $\mathbb{R}^q$
5:      Eigenstructure of new dot matrix : $\lambda_i$ and $(1/\sqrt{\lambda_i})\tilde{\mathbf{X}}^T\mathbf{u}_i \in \mathbb{R}^n$.
6:      Approximate eigenstructure of dot matrix : $\lambda_i$ and $(1/\sqrt{\lambda_i})\tilde{\mathbf{X}}^T\mathbf{u}_i \in \mathbb{R}^n$.
7:      Approximate eigenstructure of covariance matrix : : $\lambda_i$ and $(1/\lambda_i)\mathbf{X}\tilde{\mathbf{X}}^T\mathbf{u}_i \in \mathbb{R}^d$.
8:      Select $m$ as discussed above.
9:      return first $m$ eigenvectors and eigenvalues.
10: end procedure
```

5. Datasets Used in Experiments

All simulations were executed against the credit card fraud dataset, which is freely available on the Kaggle repository via the following link: https://www.kaggle.com/ datasets/mlg-ulb/creditcardfraud, accessed on 20 May 2022. This dataset consists of trans- actions generated by credit cards in Europe in September 2013 during the time span of two days. The dataset represents a binary classification challenge composed of only two target variables (classes)—the positive class, which denotes fraudulent transactions and the negative class that represents regular transactions. Moreover, the dataset is extremely asym- metrical (imbalanced) containing only 492 fraud instances out of 284,807 total transactions. Therefore, the positive class (frauds) represent only 0.172% of the dataset.

In the experiments, the original credit card fraud dataset, as it is hosted on the Kaggle, is used, and the goal of these experiments was to establish how Principal Component Analysis (PCA) is one of the most widely used data analysis methods in machine learning and AI perform on highly imbalanced data. Analyze the distribution of transaction amounts. Here's a quick look at transaction amounts in Table-1 : And here's what the distribution looks like on a log-scale histogram (makes the long right-tail easier to see) in Fig-1.

Amount	
count	284807
mean	88.35
Std	250.12
Min	0
25%	5.6
50%	22
75%	77.165
max	25691

Table1Transaction amounts

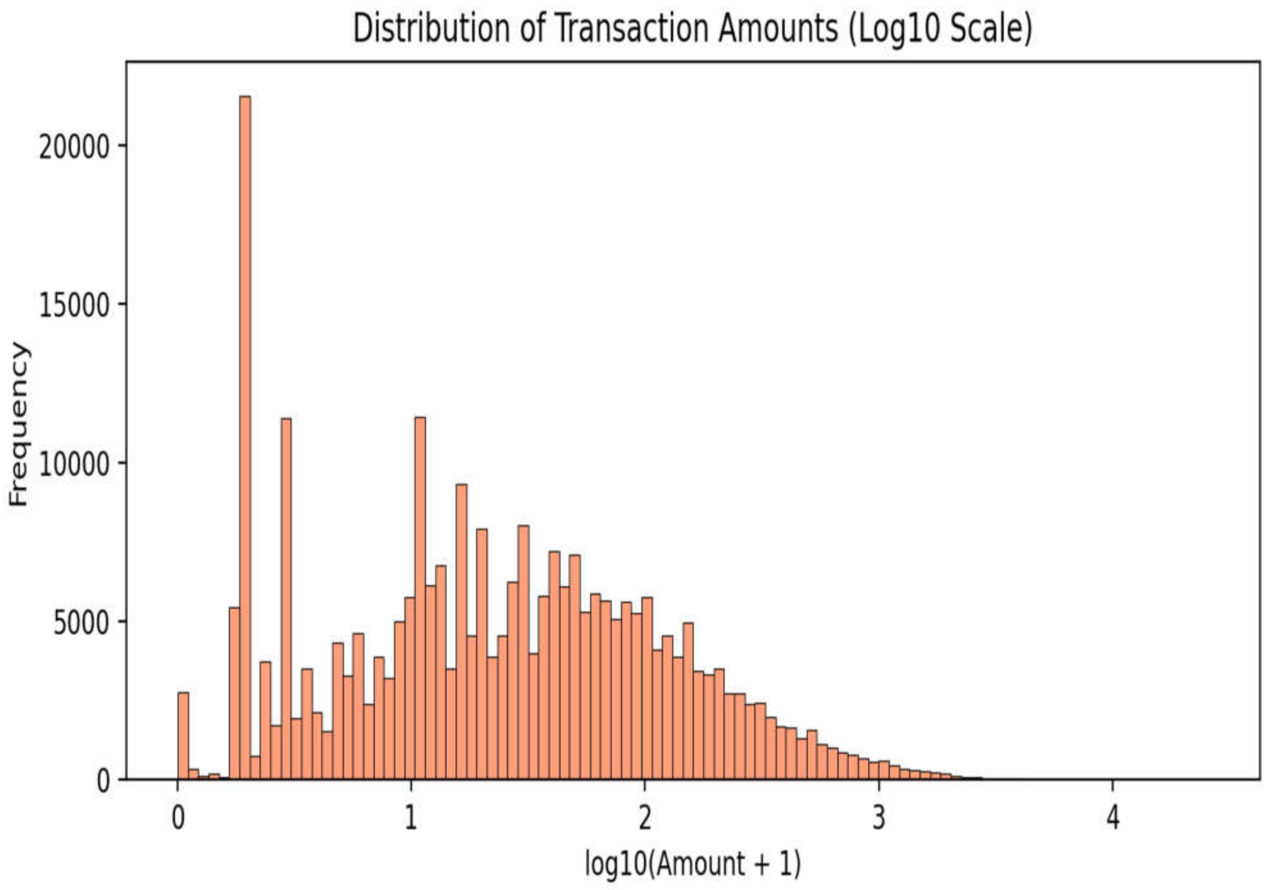


Fig-1. Distribution of transaction amount

PCA COMPONENT MATRIX

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9
V1	-0.17	0.0	-0.01	0.0	0.07	-0.06	-0.0	0.0	-0.0
V2	-0.39	-0.02	0.07	0.03	-0.11	0.05	-0.09	0.05	-0.05
V3	-0.16	0.01	-0.05	-0.04	0.18	-0.04	0.05	0.02	-0.09
V4	0.07	-0.01	0.08	0.06	-0.07	0.14	0.06	0.55	-0.16
V5	-0.29	0.01	-0.01	-0.04	0.0	-0.2	0.02	-0.37	0.08

Table-2 PCA component matrix(First 10 PC,s)

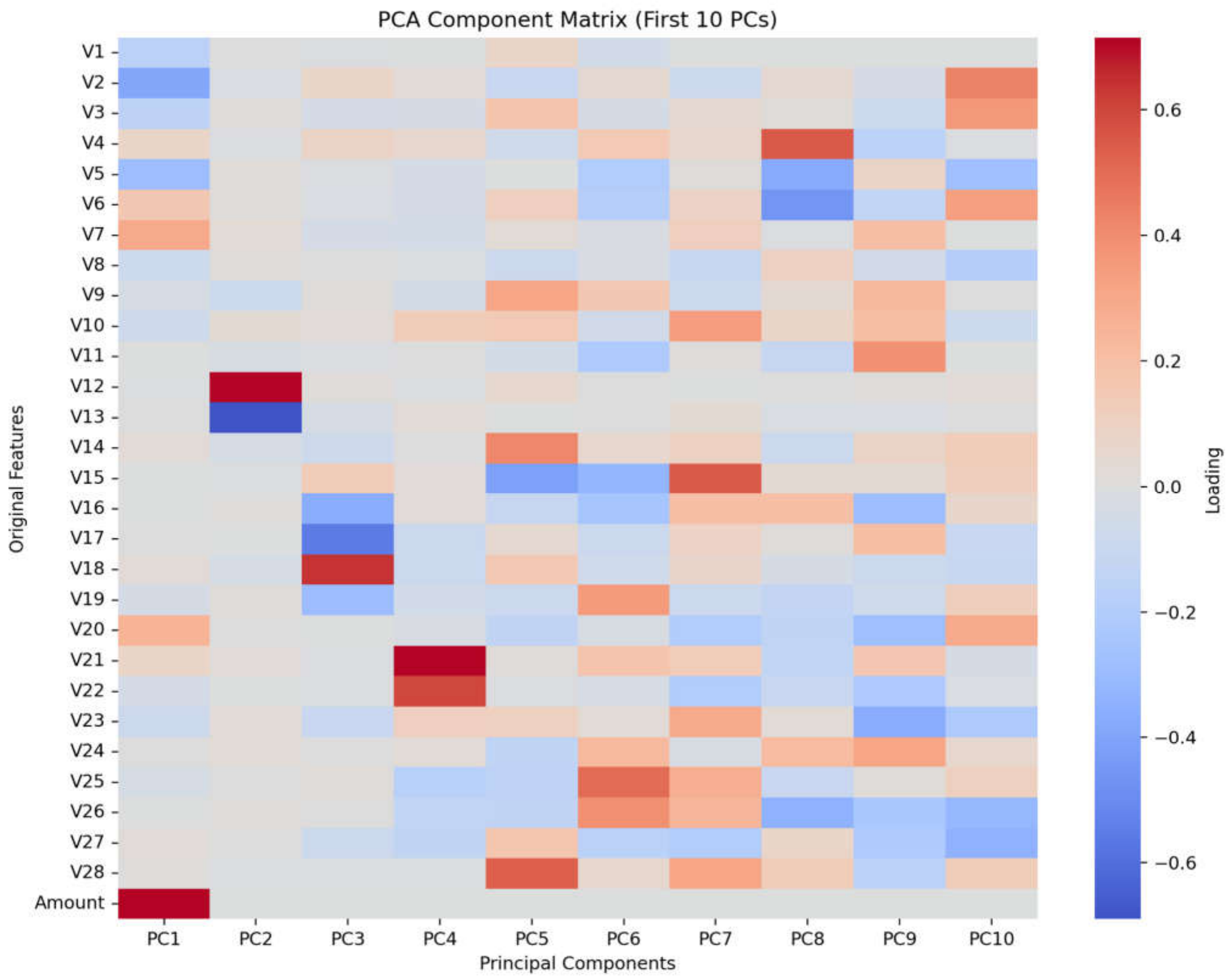


Fig-2 PCA component matrix(First 10 PC,s)

In the component matrix and heat-map Rows show the original variables (V1 … V28 + Amount) and Columns show principal components. The Cell value ("loading") show weight of that original variable in the PC; red ≈ positive influence, blue ≈ negative, white ≈ near-zero.
According to Fig-2 Key patterns you can see PC 1 is still driven primarily by Amount (strong red at bottom) and a few V-features with opposite sign (blue), PC 2 is essentially the V12 vs V13 axis (deep red vs deep blue) and PCs 3-5 pick up correlated trios such as V18 / V17 / V16 and V21-V22 vs V25-27.

PRINCIPAL COMPONENT ANALYSIS

	PC1	PC2	PC3	PC4	PC5
0	0.32370619572924497	0.25760488429210215	0.1945962954162684	0.06520773269880602	-0.47145922724151607
1	-0.47241623299580704	0.37868444929586076	-0.08481031515298265	-0.7398582264872989	-0.5009336420954141
2	1.7749788367879362	-0.35090920943032783	1.1952307428961457	1.17197540661077	-0.2772711314917756
3	0.24528273714878746	-0.25334583239440733	2.623574972166434	-0.5217077322203234	0.979751884846719
4	-0.06587600522721089	-0.491490688243078	0.08834703992258976	0.4914890724434351	0.11230087326255553

Table 3 PCA (First 5 PC,s)

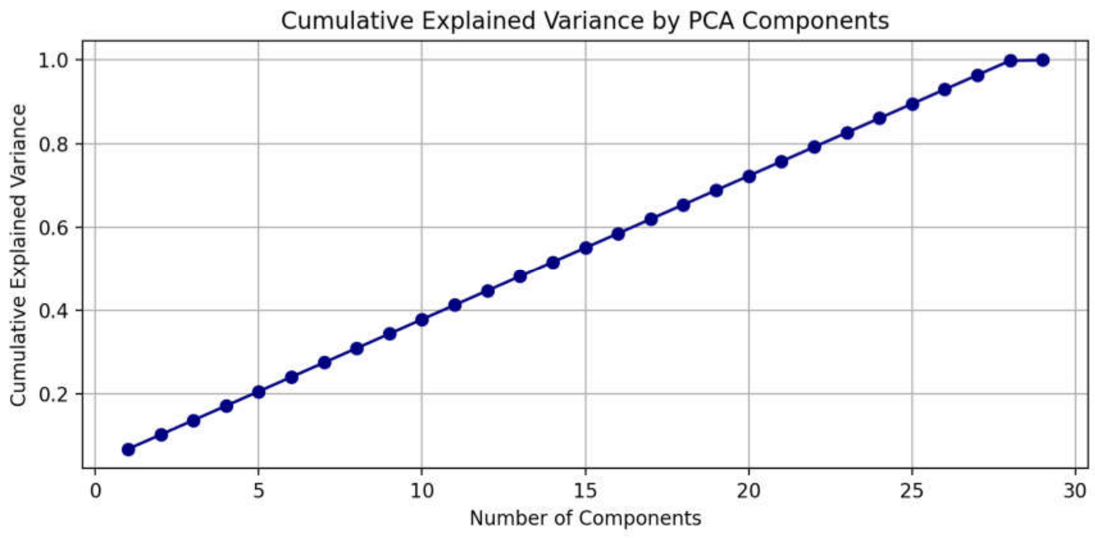


Fig. 3 Cumulative Explained Variance by PCA Components

Here's what the PCA tells us: The curve rises almost line-by-line; there isn't one or two dominant components. The Rough guide-posts: ~10 components give you ~40 % of the total variance , ~20 components reach ~70 % and You need most of the 29 components to get above 95 %. So dimensionality can be trimmed, but not drastically—information is fairly spread out.The tiny table shows the first five new axes (PC1-PC5) for a handful of transactions; they're simply the original V-features + Amount re-expressed in an orthogonal space. Next steps could be: 1. Decide a variance target (e.g., keep 15 PCs) and create a reduced dataset and Visualize fraud vs. non-fraud in PC1–PC2 space to see if separation is visible. Feed the reduced features into a classifier to check speed/accuracy trade-offs.

Identify top features contributing to PCs

	PC1	PC2	PC3	PC4	PC5
Amount	0.7071067811865479	nan	nan	nan	nan
V7	0.29330648665816017	0.03120686439734889	nan	nan	nan
V20	0.2505572472114237	nan	nan	nan	nan
V2	-0.39230119438433814	nan	nan	nan	nan
V5	-0.2852191777998456	nan	nan	nan	nan
V1	-0.16810100452217872	nan	nan	nan	nan
V12	nan	0.7132337220839206	nan	nan	nan
V10	nan	0.03957882238473481	nan	0.12673595344860233	nan
V13	nan	-0.6897240761250465	nan	nan	nan

Table 4 Top Features contributing to PC,s

In the table 4 Columns show principal components 1-5 and Rows show the original variables with the largest positive (top) and negative (bottom) loadings for each PC (only the strongest three in either direction are shown). The numbers are the coefficients in the linear combination; higher magnitude ⇒ stronger influence.

Highlights PC 1 ("transaction size" axis) , Amount, V7, V20 push scores upward and V2, V5, V1 push them downward. Interpretation: the first component contrasts raw Amount (and a couple of related anonymised features) against several V-features; large positive PC1 ≈ big purchases , PC 2 ("time-shift vs. profile" axis) and V12 dominates the positive side while V13 dominates the negative, giving a strong one-dimensional contrast between those two latent features.

PC 3 ("balance-oscillation" axis) V18 drives positive values, while V17 and V16 drive negatives; this pair of correlated features is being teased apart. PC 4 ("V21/V22 against V25-27 cluster") Positive: V21 and V22 and Negative: V25, V27, V26. PC 5 ("high-frequency tail"), Positive: V28, V9, V14 and Negative: V15 and V25.

Next steps you might consider keep the top n PCs and visualise fraud-vs-legit separation, Use these loadings to understand which engineered features matter most for a model. Rotate or varimax-spin the components for even clearer interpretation.

Compare PCA results before and after scaling

	PC	Unscaled_Var%	Scaled_Var%
0	1	99.95	6.75
1	2	0.01	3.45
2	3	0.0	3.45
3	4	0.0	3.45
4	5	0.0	3.45
5	6	0.0	3.45

Table 5 PCA results before and after scaling

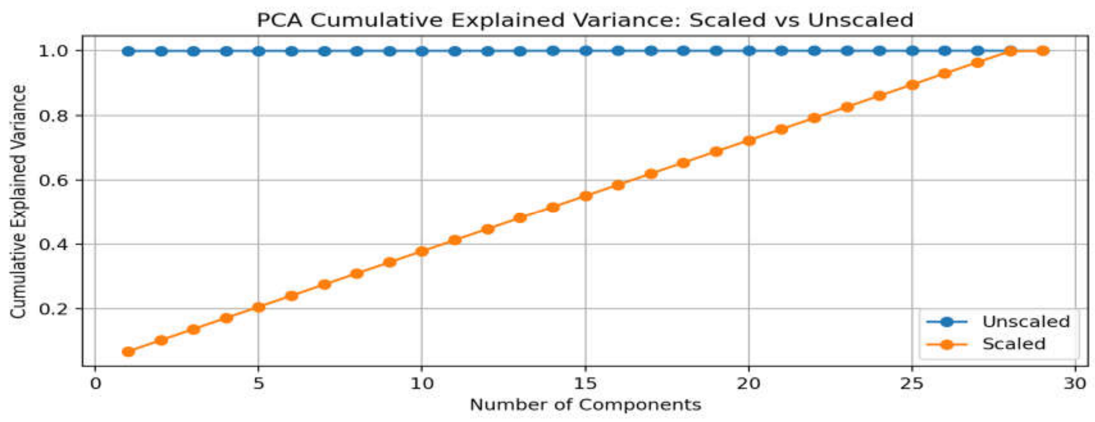


Fig 4 PCA results before and after scaling

In fig 4 Un-scaled PCA, The raw Amount column dominates the variance (≈ 100 % in PC1),Every other component explains almost nothing and Result: one "size" axis, little insight elsewhere. Scaled PCA, Standardising evens the playing field, so each V-feature can contribute and Variance is now spread more evenly (≈ 6-7 % per component). • You need ~15 PCs to reach 60 % cumulative variance, but you gain a multidimensional signal instead of a single "big transaction" axis.

PCA SCORES

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9
0	0.32370619572924497	0.25760488429210215	0.1945962954162684	0.06520773269880602	-0.47145922724151607	-0.11216689137467226	0.5432992523095345	0.5669331587819579	-0.2876604333919832
1	-0.47241623299580704	0.37868444929586076	-0.08481031515298265	-0.7398582264872989	-0.5009336420954141	-0.628500815781507	0.8352779025554086	0.07701696158938355	0.19872582299677097
2	1.7749788367879362	-0.35090920943032783	1.1952307428961457	1.17197540661077	-0.2772711314917756	-2.288464091905241	1.4962474013655271	-0.9753214864934595	0.0885763065388899
3	0.24528273714878746	-.25334583239440733	2.623574972166434	-0.5217077322203234	0.979751884846719	-0.5827427077774181	0.036587095499876425	-1.21636988790352	-0.8339107541567488
4	-0.06587600522721089	-0.491490688243078	0.08834703992258976	0.4914890724434351	0.11230087326255553	0.8692712829027893	-0.08472348162334035	-0.045673100536470196	-0.8759389891195877

Table 6 PCA Scores

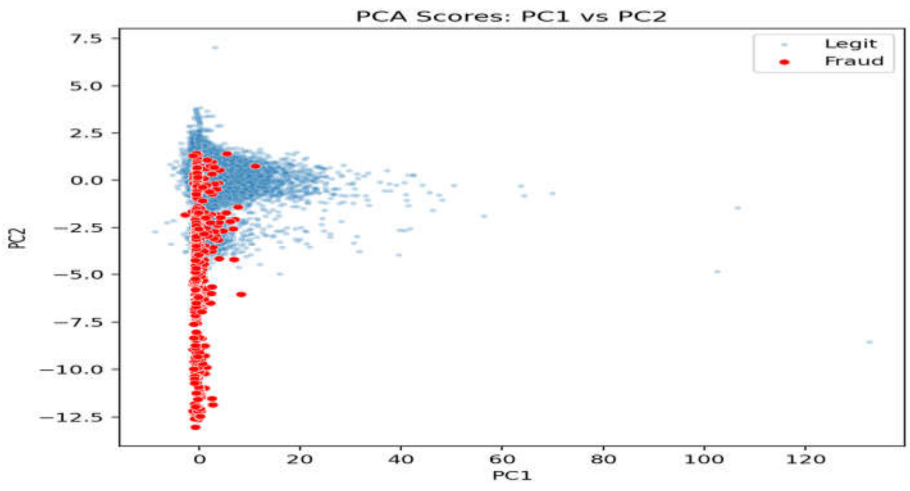


Fig 5 PCA Score PC1 vs PC2

These are the PCA scores—each row is a transaction, and each "PC k" column is that transaction's coordinate on the k-th principal component (after scaling all features). I kept enough PCs to capture ~95 % of the variance (27 components here). The scatter-plot shows the first two scores, with fraud cases in red. You can already see good separation along PC 1 / PC 2.

Visualize PC scores distribution

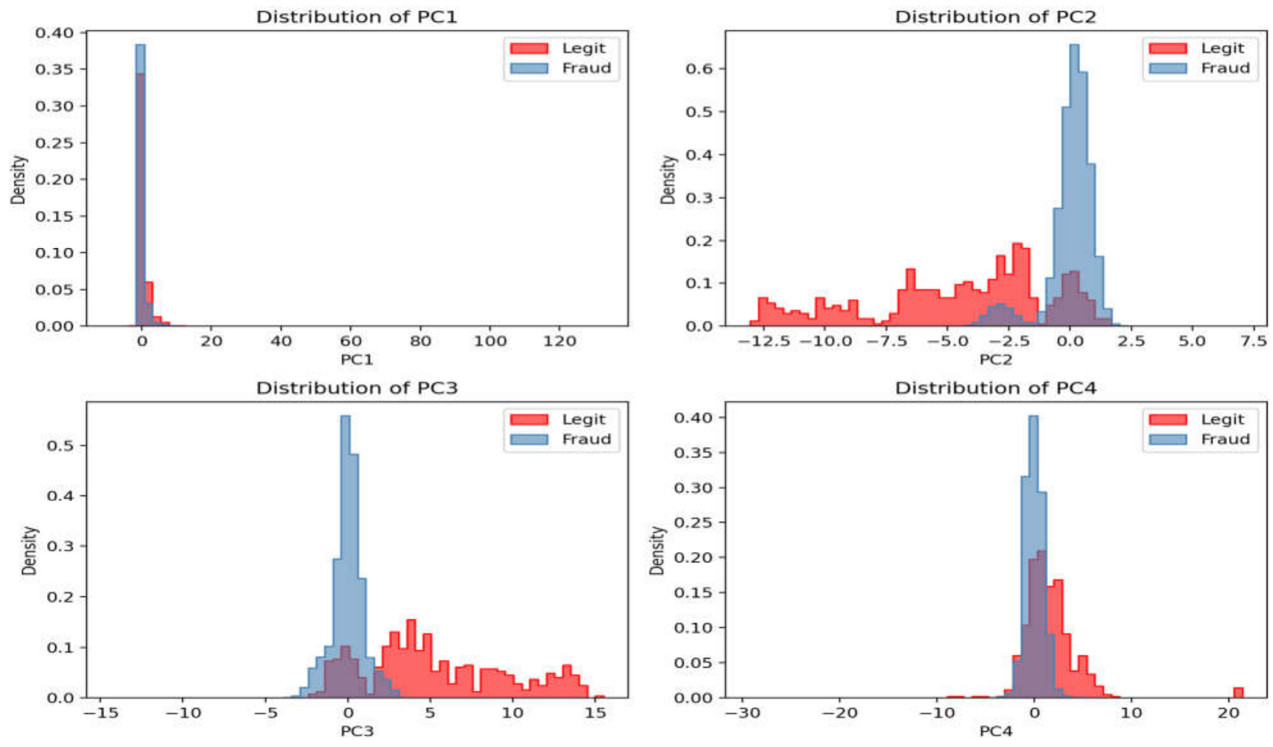


Fig 6 PC Score distibution

Here's how the first four principal-component scores are distributed for legitimate (blue) vs. fraud (red) transactions:
PC 1 Mostly separates records by overall transaction size—both classes are tightly centered near zero except for a handful of very large legitimate transactions (the long right-hand tail). PC 2 Shows clear class separation: fraud cases cluster around 0, while legitimate ones are shifted left. This mirrors the strong V12-V13 contrast we saw in the loading matrix. PC 3 Again the two groups occupy distinct ranges, with fraud concentrated in a narrow negative band and legitimate transactions spreading out toward positive values. PC 4 Distributions still differ, though overlap starts to increase. PCs beyond this point add finer-grained contrasts.
Overall, PCs 2–4 already provide useful univariate discrimination, which is why a simple scatter of PC 1 vs PC 2 showed the red points standing apart.

PCA SCREEN PLOAT

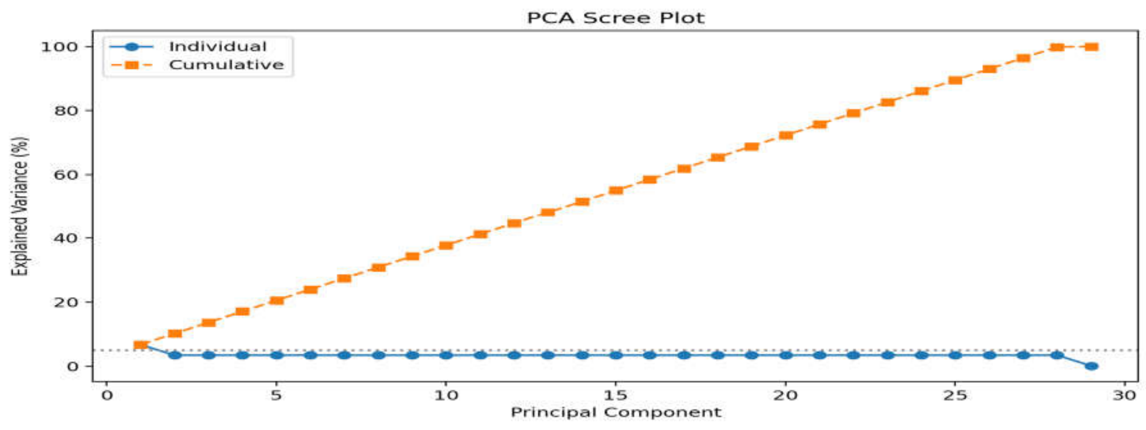


Fig 7 PCA Screen plot

Above is the PCA scree plot: Blue circles show individual-component variance, No single PC dominates; each explains < 5 % of the total, and Orange squares show cumulative variance. It climbs roughly linearly, crossing 70 % at ~20 PCs and 95 % at 27 PCs (the cut-off we used for the scores).
Interpretation, Because variance is so evenly spread, dimensionality reduction is feasible but you need quite a few components to retain most information. In practice, keeping the top 15–20 PCs already preserves the bulk of the structure while slashing dimensionality from 29 variables to a more tractable size.

PCA BIPLOT

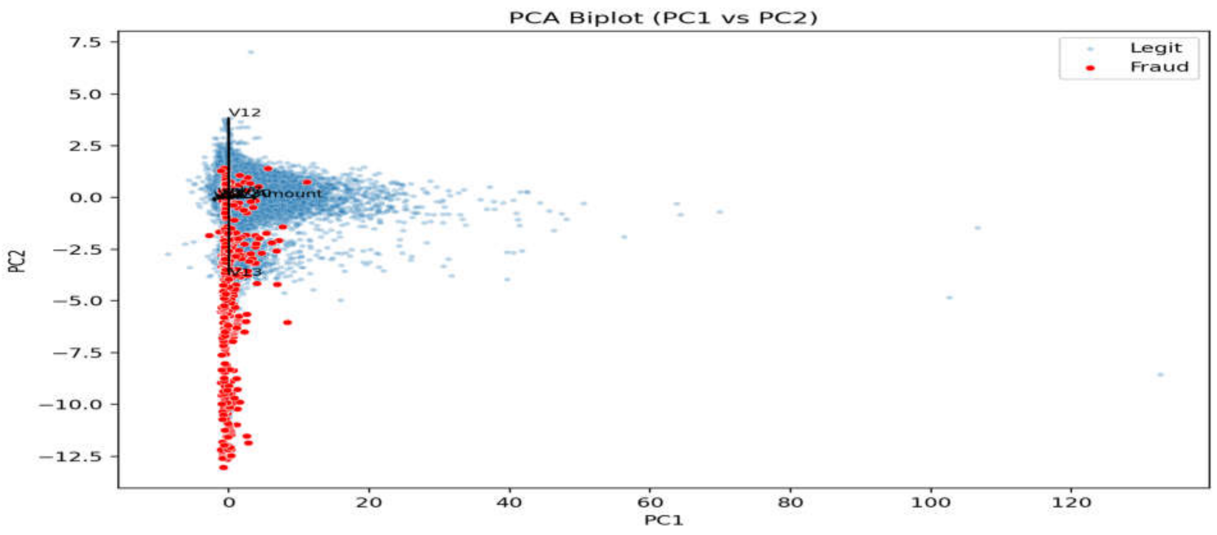


Fig 8 PCA Biplot

The biplot overlays two things on the PC1–PC2 scatter: Scores (points) Blue show legitimate, Red show fraud. Fraud points hug the negative-PC1 axis, confirming the separation we saw earlier. Loadings (black arrows), The arrows show which original variables influence each PC direction. For example, V12 points upward along PC2, while V17 and V10 lean left along PC1—so transactions with extreme V17/V10 values drive large negative PC1 scores (where fraud clusters).

In short: PC1 is mainly a contrast driven by V17, V10, and related features, while PC2 is steered by V12. Together they set apart the fraud region (lower-left quadrant) from the bulk of normal transaction.

	PC1	PC2	PC3	PC4	PC5
V1	-0.23521185934521036	0.004095075197121527	-0.008747471628066225	6.131379507616363e-16	0.07412517733349201
V2	-0.5489193453469995	-0.015285882453687814	0.07437535894161647	0.025517664454767837	-0.10569209230380334
V3	-0.21782917801302595	0.011620736796068386	-0.04719114101977063	-0.03799514382749516	0.1811685508407297
V4	0.10198496417164304	-0.012977055284778084	0.08417871900865681	0.06122505271950301	-0.06590051417337967
V5	-0.39908704485084034	0.01289403968507728	-0.01117156713632699	-0.04497507310849639	0.0009110923604312544
V6	0.22309795579736147	0.016228499275178384	-0.017258649062433663	-0.043054472738186335	0.11616948661229475
V7	0.41040304477046036	0.031206864397348898	-0.04686653912235015	-0.04872927320186026	0.032650883550410965
V8	-0.10647564523839272	0.010299862783005554	0.004098800425593699	-0.0075881634982455275	-0.09365247702167853
V9	-0.045703534636656065	-0.07761854036476451	0.012206684029195315	-0.0482745762212936	0.3108981801692219

Table 7 Correlation Between Original and PCA Components

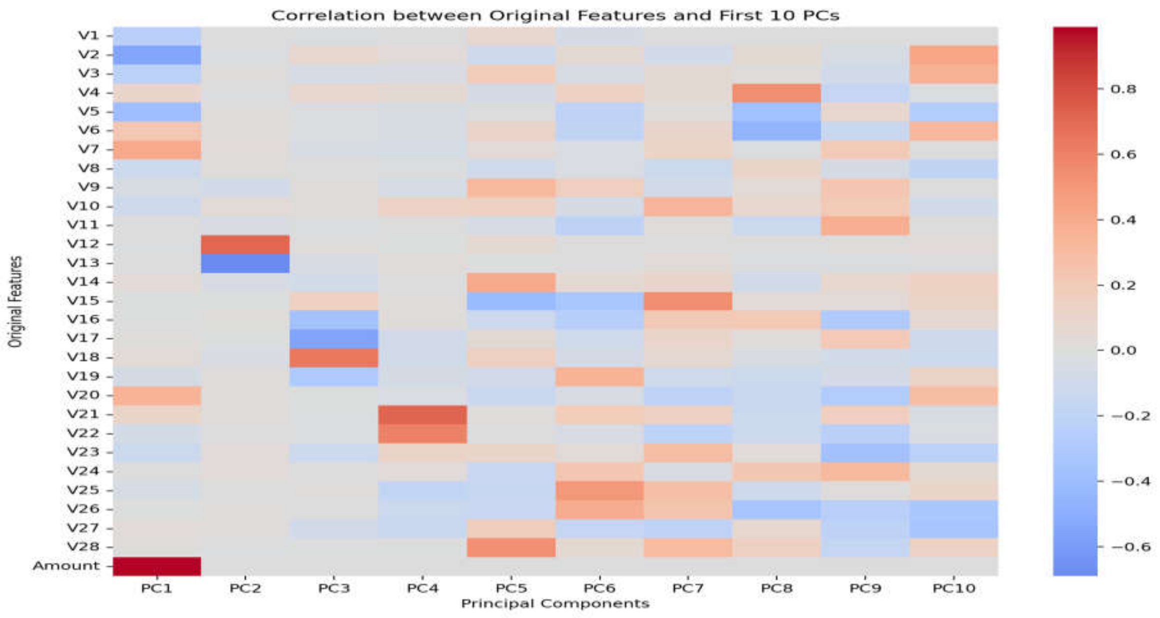


Fig 9 Correlation between original and PCA component

The tiny table gives a numeric glimpse—here you see how each of the first five PCs correlates with the first ten original (scaled) features. The full heat-map then expands that view to the first ten PCs versus every original variable: Bright red / blue indicates strong positive / negative correlation. PC 1 is dominated by the Amount feature (deep red at bottom) plus negative ties to variables like V2, V5. PC 2 swings strongly with V12 and against V13, matching what we saw in the loading plot. • Later PCs (PC 4–PC 7) pick up more targeted patterns—e.g., V15 for PC 6, V18 for PC 3—showing that information is dispersed rather than concentrated in one or two axes. These correlations help interpret PCs: if a model uses (say) PC 2 heavily, we now know it's effectively leveraging the V12 ↔ V13 contrast. Let me know if you'd like deeper dives—e.g., exact coefficients for all 29 PCs or zooming into any particular variable.

6. Conclusions

For years, researchers have been interested in credit card fraud detection, and it will continue to be an attractive topic of research in the future. This is primarily due to the fact that fraud tendencies are always changing. In this study, we reviewed the classical PCA method and the problem that may arise when applying it to very small or very large high-dimensional datasets. We have also dis- cussed different methods that may be used to handle these cases. We have shown that a good PCA approximation could be achieved.
The dataset is composed of 30 numerical features, where attributes *v*1, *v2*, . . . *v*28 are obtained by applying the principal component analysis (PCA), while *v*29 and *v*30, which represent time and amount, respectively, were not transformed with the PCA. The time refers to the number of seconds elapsed between the first and each other transaction in the dataset, while the amount is the value of every transaction.
This work gives the technique for credit card fraud analysis depending on the PCA. PCA is used for extracting multiple features at a time from real-world datasets. Each PC represents an optimized value of the original attribute. Experimental results show efficient analysis of associated attribute relation. Furthermore, the proposed approach shows an optimized cluster representation effectively. This work gives the technique for credit card fraud analysis depending on the PCA. PCA is used for extracting multiple features at a time from real-world datasets. Each PC represents an optimized value of the original attribute. Experimental results show efficient analysis of associated attribute relation. Furthermore, the proposed approach shows an optimized cluster representation effectively

7. References

[1] Adelomou PA, Fauli DC, RibÃÃÃ,â€šÆÃ,â€™Â EG, Vilasis-Cardona X (2022) Quantum case-based reasoning (qCBR). Artif Intell Rev, pp 1–27 Barla A, Odone F, Verri A (2003) Histogram intersection kernel for image classification. In: Proceedings 2003 International Conference on Image Processing (Cat. No.03CH37429), vol 3, pp 3–513

[2] Bertsekas DP (1996) Constrained optimization and lagrange multiplier methods. Athena Scientific, Belmort Blum A, Hopcroft J, Kannan R (2017) Foundations of data science

[3] Boughorbel S, Tarel JP, Boujemaa N.(2005) Generalized histogram intersection kernel for image recogni- tion. In: IEEE international conference on image processing 2005, vol 3, pp 3–161

[4] Brin S, Page L (1998) The anatomy of a large-scale hypertextual web search engine. Computer Networks and ISDN Systems. 30(1):107–117. Proceedings of the Seventh International World Wide Web Conference

[5] Charikar M, Chen K, Farach-Colton M(2002) Finding frequent items in data streams. In: Proceedings of the 29th International Colloquium on Automata, Languages and Programming. ICALP '02, Springer, London, pp 693–703

[6] Cormode G, Muthukrishnan SM (2005) An improved data stream summary: the count-min sketch and its applications. J Algorithms 55(1):58–75 Fukunaga K (1990) Introduction to statistical pattern recognition. Academic Press, 2 edn, 10

[7] Gewers FL, Ferreira GR, de Arruda HF, Silva FN, Comin CH, Amancio DR, da Luciano FC 2018) Principal component analysis: anatural to data exploration. *CoRR*, abs/1804.02502 Gene H (1996) Golub CF. The John Hopkins University Press, Van Loan. Matrix Computations

[8] Hein Ma, Bousquet O (2004) Hilbertian metrics and positive definite kernels on probability measures. Tech- nical Report 126, Max Planck Institute for Biological Cybernetics, Tübingen, 2004

[9] Hein Matthias BO (2005) Hilbertian metrics and positive definite kernels on probability measures. In: RG Cowell and Z Ghahramani (eds), *Proceedings of the Tenth International Workshop on Artificial Intel- ligence and Statistics, AISTATS 2005, Bridgetown, Barbados, January 6-8, 2005*. Society for Artificial Intelligence and Statistics

[10] Hofmann T, Schölkopf B, Smola AJ (2008) Kernel methods in machine learning. Ann Stat 36(3):1171–1220 Jaakkola TS, Diekhans M, Haussler D (1999) Exploiting generative models in discriminative classifiers. In: Proceedings of the 1998 Conference on Advances in Neural Information Processing Systems II, pp487–

493, Cambridge: MIT Press

[11] Jacob CGJ (1846) Über ein leichtes Verfahren, die in der Theorie der Säkularstörungen vorkommenden Gleichungen numerisch aufzulösen. Crelle's J (in German) 30:51–94 Kumar S, Mohri M, Talwalkar A (2012) Sampling methods for the nyström method. J Mach Learn Res 13(1):981–1006

[12] Lodhi H, Saunders C, Shawe-Taylor J, Cristianini N, Watkins C (2002) Text classification using string ker- nels. J Mach Learn Res 2:419–444

[13] Marukatat S (2016) Kernel matrix decomposition via empirical kernel map. Pattern Recogn Lett 77:50–57 Petersen KB, Pedersen MS (2012) The matrix cookbook Ringnér M (2008) What is principal component analysis? Nat Biotechnol 26:303–304

[14] Schölkopf B, SchÃƒÂ¶lkopf B, Burges CJ, Smola AJ (eds) (1999) Advances in Kernel methods: support vector learning. MIT Press, Cambridge

[15] Bernhard S, Sebastian M, Burges Christopher JC, Phil K, Klaus-Robert M, Gunnar R, Smola Alexan- der J (1999) Input space versus feature space in kernel-based methods. IEEE Trans Neural Netw 10(5):1000–1017

[16] Schölkopf B, Smola AJ, Bach F (2018) Learning with Kernels: support vector machines. optimization, and beyond. The MIT Press, Regularization Schölkopf B, Smola AJ, Müller K-R (1999) Kernel principal component analysis. MIT Press, Cambridge Shawe-Taylor J, Cristianini N (2004) Kernel methods for pattern analysis. Cambridge University Press,Cambridge

[17] Shi Q, Petterson J, Dror G, Langford J, Smola A(2009) Hash kernels. In DA Van Dyk and M Welling, (eds), Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, AISTATS 2009, Clearwater Beach, April 16–18, 2009, volume 5 of *JMLR Proceedings*, pp 496–503. JMLR.org Sirovich L, Kirby M (1987) Low-dimensional procedure for the characterization of human faces. J Opt Soc Am A.

[18] The orl database of faces. http://www.face-rec.org/databases/ Turk M, Pentland A (1991) Eigenfaces for recognition. J. Cognit Neurosci 3(1):71–86 Wang M, Deng W (2018) Deep face recognition: a survey. *CoRR*, abs/1804.06655

[19] Weinberger K, Dasgupta A, Langford J, Smola A, Attenberg J (2009) Feature hashing for large scale mul- titask learning. In: Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09, pp 1113–1120, New York: ACM

[20] Williams CKI, Seeger MW (2000) Using the nyström method to speed up kernel machines. In: TK Leen, TG Dietterich, V Tresp, (eds), Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, pp 682–688. MIT Press.

[21] M. Kirby and L. Sirovich, "Application of the Karhunen-Loeve Procedure for the Characterization of Human Faces," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 12, no. 1, pp. 103-108, Jan. 1990. M. Kirby and L. Sirovich, "Application of the Karhunen-Loeve Procedure for the Characterization of Human Faces," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 12, no. 1, pp. 103-108, Jan. 1990.

[22] A. M. Mart´ınez, "Recognition of Partially Occluded and/or Imprecisely Localized Faces Using a Probabilistic Approach," *Proc. Computer Vision and Pattern Recognition,* vol. 1, pp. 712-717, June 2000.

[23] B. Moghaddam and A. Pentland, "Probabilistic Visual Learning for Object Representation," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 19, no. 7, pp. 696-710, July 1997.

[24] H. Murase, F. Kimura, M. Yoshimura, and Y. Miyake, "An Improvement of the Autocorrelation Matrix in Pattern Matching Method and Its Application to Handprinted 'HIRAGANA'," *Trans. IECE Japan,* vol. 64D, no. 3, 1981.

[25] H. Murase and S. Nayar, "Visual Learning and Recognition of 3D Objects from Appearance," *Int'l J. Computer Vision,* vol. 14, pp. 5-24, 1995.

[26] S.K. Nayar, N.A. Nene, and H. Murase, "Subspace Methods for Robot Vision," *IEEE Trans. Robotics and Automation,* vol. 12, no. 5, pp. 750-758, 1996.

[27] L. Sirovich and M. Kirby, "A Low-Dimensional Procedure for the Characterization of Human Faces," *J. Optical Soc. Am. A,* vol. 4, no. 3, pp. 519-524, 1987.

[28] M. Turk and A. Pentland, "Eigenfaces for Recognition," *J. Cognitive Neuroscience,* vol. 3, no. 1, pp. 71-86, 1991.

[29] J.J. Weng, "Crescepton and SHOSLIF: Towards Comprehensive Visual Learning," *Early Visual Learning,* S.K. Nayar and T. Poggio, eds., pp. 183- 214, Oxford Univ. Press, 1996.