CourseWork 1: Scratch Implementation of PCA

This coursework accounts for 30% of the total mark for the module, 6CS012 (Artificial Intelligence & Machine Learning)

Read the instructions carefully and do whatever necessary. Your original ideas are highly appreciated :)

Student Name:Sunil Ghimire Student ID: 1928584

Read about the 'Principal Component Analysis'.
Recommended Video: https://www.youtube.com/watch?v=FgakZw6K1QQ (https://www.youtube.com/watch?v=FgakZw6K1QQ)
Online Course (Coursera): https://www.coursera.org/learn/pca-machine-learning (https://www.coursera.org/learn/pca-machine-learning)

What is "Dimensionality Reduction Problem"? Why is it necessary in Machine Learning?

Ans:

Machine learning is nothing but a research area that allows machines to “read” like humans without any need for specific programming.

Predictive modelling is known as predictive analysis that attempts to forecast future events or outcomes by evaluating trends that are likely to predict future outcomes. The aim of predictive modelling is to address these questions: “What will most possibly happen in the future, based on observed historical behaviour”?

After the concept of machine learning and predictive modelling, there is another term known as dimensionality reduction. So, problems with classification in machine learning often involve too many factors on the basis of which final classification is done and these factors are basically variables called features and if there is a higher number of features in the dataset then the more difficult it gets to visualize and work on the training set. Sometimes, most of these features present in the given dataset are correlated with each other and therefore redundant. So, this is the main reason that algorithms of dimensionality reduction come into action to avoid redundant and correlation of features in the dataset. Thus, in statistics, machine learning and information theory, dimensionality reduction or dimension reduction refers to the techniques that can reduce the number of input variables in a dataset. In another word, dimensionality reduction is the process of reducing the number of random variables under consideration by obtaining the set or principal variables More input variables in the dataset often make a predictive modelling task more challenging or difficult handle model which is more generally referred to as the curse of dimensionality. While dimensionality reduction techniques are often used for data visualization on high-dimensionality statistics, such techniques can be used in applied machine learning to simplify a dataset of classification or regression to match a predictive model better.

Dimensionality reduction approaches can be divided into feature selection and feature extraction.

Feature selection approaches try to find features or attributes of the dataset also called the subset of the input variables The three strategies are The filter strategy (Example: Information Gain) The wrapper strategy (Example: Search Guide by accuracy) The embedded strategy (Example: Selection features or removed features while building the model based on prediction error.) Similarly, feature extraction is also known as feature projection transforms the data from a high dimensional space into lesser dimensional space. Data transformation may be linear as in the case with principal component analysis (simply called PCA), but there are also many techniques of non-linear dimension reduction. Tensor representation may be used for multidimensional data in the reduction of dimensionality by multilinear subspace learning.

An intuitive example of dimensionality reduction can be discussed through a simple fraud classification problem, whether we need to classify whether the credit card is fraud or genuine. This can involve a large number of features, such as time, amount of money, type of account (saving or current account) and many more features. Any of these features can therefore overlap. In another case, a classification issue that relies on both humidity and rainfall that collapse into only one underlying feature, because both of these are strongly correlated. So, in such problems, we can reduce the number of features. It might sound trivial but is an essential part of the machine learning model. Too much dimension acts as noise to our machine learning model, hence it is vital in many cases to use dimension reduction.

1. More dimension may function as a noise rather than a feature, and the performance value might not be connected to any aspect. Hence, decision making is all random weight.
2. Less number of dimensions implies less number of parameters to learn and hence results in a faster and efficient model generally. Sometimes a huge number of dimensions cause noise and the model learns nothing.
3. In tabular data, when the column decreases, then the number of rows needed to learn is also decreased in most cases.
4. Two-dimensional mapping is also used to simplify the representation of high dimensional datasets.
5. When the dimensions can be interpreted meaningfully, projection can be used along that dimension to explain certain behaviours.

Several techniques involve in dimension reduction are listed below:

1. Dimension reduction by feature elimination
2. Dimension reduction by Matrix Factorization
3. Dimension reduction by projection-based techniques.

Advantage of Dimensionality Reduction are listed below:

1. It helps in data compression and hence reduced storage space.
2. It reduces the computation time.
3. It also helps to remove redundant features.

Disadvantages of Dimensionality Reduction are listed below:

1. It may lead to some amount of data loss.
2. PCA trends to find linear correlations between variables, which is sometimes undesirable
3. PCA fails in cases where mean and covariance are not enough to define datasets.
4. We may not know how principal components to keep - in practice, some thumbs rules are applied.

Methods of dimensionality reduction are listed below:

1. Principal Component Analysis (PCA)
2. Linear Discriminant Analysis (LDA)
3. Generalized Discriminant Analysis (GDA)

What is PCA? Discuss few applications in Machine Learning...

Ans: PCA which is an unsupervised and non-parametric statistical technique primarily used to minimize the

dimensionality of a dataset of many variables correlated with each other by maximum retention of variation present in the dataset.

The variables are transformed into a new set of variables called principal components and orthogonally ordered such that the retention of variation present in the original variables decreases as we move down in the order, this was 1st principal component (PC1) has maximum variation. In this way, PCA is used in machine learning for predictive models and data analysis. Also, some of the reason how PCA is used in machine learning are listed below:

1. We can visualize the large complex data in lower-dimensional space.
2. We can eliminate the unnecessary features. So, we can reduce the original feature space to a lower-dimensional space that greatly reduces the computing resources.
3. We can use it as a feature selection technique
4. We can use the principal components as input for supervised learning problems.
5. In machine learning, it helps to solve the curse of dimensionality problem. If n-sample is dense enough for 1D (only one feature0), then in dd dimension we need n^d samples. n^d grows really fast as a function of d. We can reduce the dimension of data using PCA.
6. PCA is a basic method of studying any data type like time series, photographs (eigenfaces), etc. This offers an "explanation" of an input-data focused on the key components and on the classification that you may do.

Write Short notes on the following topics:

a) Variance & Covariance

Variance

Variance (σ^2) is statistics is a measurement of the difference numbers in a data collection. Thus, it calculates how far each number in the set is from the mean and therefore from every other number in the set. Note:

1. Variance is used to compare the relative performance of difference number in a data collection
2. Because the results can be difficult to understand, standard deviation is often used instead of variance.
3. In either case, the goal for the investor is to improve asset allocation.

Variance is calculated by taking differences between each number in the data set and the mean, then squaring the differences to make positive, and finally dividing the sum of the squares by the number of values in the data set.

The formula for variance is Variance (σ^2) = $\frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n}$

Where,

xi = the ith data point

x = the mean of all data points

n = the number of data points

The square root of the variance is the standard deviation (σ). Variance measures variability from the average or mean. Variability is the volatile or variance is the risk factor. The variance statistics can thus help to determine the risk that an investor assumes when purchasing a specific security. The large variance

means that the numbers in the sample are far from the mean and from each other, while the reverse implies a low variance. Variance can be negative. A variance with the value 0 indicates that all values within a series of numbers are the same. And all the variance that are not zero will be positive numbers.

The advantage of variance is that any deviation from the mean is handled the same irrespective position. The squared variations can not add up to zero, providing the impression of no variations in the results at all.

The disadvantages of variance is that it is not easily interpreted. Users of variance often employ it primarily to take the square root of their value which indicates the standard deviation of the dataset.

Covariance

The measure of the simultaneous variation between the random variables X and Y is called co-variance, written by $cov(x, y)$ or σ_{xy}The covariance of two statistically independent random variables X and Y is zero. The converse, however, is not generally true i.e. Two variables may have zero covariance and still not be statistically independent. Covariance is calculated by analyzing at-return surprises (standard deviations from the expected return) or by multiplying the correlation between two variables by the standard deviation of each variable.

Note:

1. Covariance is the statistical method used to evaluate the relation between movement of two independent variables.
2. When the two independent variables tend to move together, they are seen as having a positive covariance and when they move inversely the covariance is negative.
3. Risk and volatility can be reduced in two independent variables that have a negative covariance.

Covariance can be calculated using 5 variables from the data. They are:

x_i = a given x value in the dataset

x_m = the mean, or average of the x values

y_i = the y value in the dataset that corresponds with x_i

y_m = the mean, or average of the y values

n = the number of data points. Form the above information, the formula of covariance is

$$cov(x, y) = \sum (x_i - x_m)(y_i - y_m)$$

b) Eigenvalues & Eigenvectors

Eigenvector

An eigenvector is a vector whose direction remains unchanged when linear transformation is applied to it. In another word, eigenvectors are unit vectors, which means that their length or magnitude is equal to 1 that is often referred as right vectors which simply means the column vectors (as opposed to a row vector or a left vector)

Eigenvalues

An eigenvalue is a special set of scalars associated with a linear system of equations (i.e. a matrix equation) that are sometimes known as characteristic roots. In another word, eigenvalues are coefficients applied to eigenvectors that give the vectors their length or magnitude. For example, negative eigenvalue may reverse the direction of the eigenvector as part of scaling it.

For example

$$\begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad = 2 \begin{bmatrix} 1 \\ -4 \end{bmatrix}$$

which is in the form Ax = λx

where x--> eigenvector(proper vector)

y--> eigenvalues(proper value)

Given; A = $\begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$

$A - \lambda I$ = $\begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

= $\begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}$

= $\begin{bmatrix} 3-\lambda & 1 \\ 1 & 3-\lambda \end{bmatrix}$

Now, we know

$(A - \lambda I)x = 0$

$|A - \lambda I| = 0$ is characteristics of matrix where λ is eigenvalue of A

$\begin{vmatrix} 3-\lambda & 1 \\ 1 & 3-\lambda \end{vmatrix} = 0$

= $(3-\lambda)^2 - 1 = \lambda^2 - 6\lambda + 8$

Therefore, λ_1 = 4 and λ_2 = 2

When λ_1 = 4, then

$(A - \lambda I)x = 0$

= $\begin{bmatrix} 3-\lambda & 1 \\ 1 & 3-\lambda \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

= $\begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

or, $-x + y = 0$
or, $x - y = 0$

so, $x = y$

The eigenvector for eigenvalue(λ_1 = 4) is $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$

When λ_2 = 2

= $\begin{bmatrix} 3-\lambda & 1 \\ 1 & 3-\lambda \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

= $\begin{bmatrix} 3-2 & 1 \\ 1 & 3-2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

= $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

So, we get $x + y = 0$ and $x + y = 0$

which means $x = -y$ The eigenvector for eigenvalue(λ_2 = 2) is $\begin{bmatrix} 1 \\ -1 \end{bmatrix}$

So, for each eigenvalue λ. Now, every system will have infinetely value solutions beacause if x is the eigenvector. So is and multiple of x. So, our strategy will be try to find the eigenvector with x = 1 and necessary. We can scale up the value. If there is no such eigenvector, we know that x must in fact be zero, and we instead look for the eigenvector with y = 1 and so on.

c) Eigen-value decomposition

Matrix decompositions are a more useful technique to simplify a range of more complicated operations for reducing a matrix to their constituent parts which analyzes the structure of the matrix. Even the eigen-decomposition does not exist for all square matrices, it has a particularly simple expression for a class of matrices often used in multivariate analysis such as correlation, covariance, or cross-product matrices So, the most powerful used matrix decomposition is known as eigenvalue decompositions that decompose a matrix into eigenvectors and eigenvalues and such type of decomposition also plays an important role in machine learning as in principal component analysis method (PCA).

The eigen-decomposition is important because it is involved in problems of optimization. Problems of optimization can be easily solved using eigenvalue decomposition. For example, in PCA, we want to analyze an I × J matrix X where the rows are observations and the columns are variables describing these observations. The goal of analysis is to find row factor scores, such that these factor scores “explain” as much of the variance of X as possible, and such that the sets of factor scores are pairwise orthogonal. The eigenvectors of a matrix is given below

$$Ax - \lambda Ix = 0$$

$$(A - \lambda I)x = 0 \text{ .....(ii)}$$

where I is the identity matrix. The values of λ holds the eigenvalues of A. It turns out that the equation is equivalent to:

$$det(A - \lambda I) = 0, \text{ ..... (iii)}$$

where $det()$ is the determinant of a matrix.

Let's see the eigendecomposition for the matrix: $\begin{bmatrix} 1 & 0 \\ 1 & 3 \end{bmatrix}$

From (3)

$$\begin{vmatrix} 1-\lambda & 0 \\ 1 & 3-\lambda \end{vmatrix} = 0$$

$$(1 - \lambda)(3 - \lambda) = 0$$

we get directly, $\lambda_1 = 1$ and $\lambda_2 = 3$. The above expression is usually referred as characteristic equation of matrix.

Plugging λ_1 into (1), we get:

$$= \begin{bmatrix} 1 & 0 \\ 1 & 3 \end{bmatrix} \quad \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \quad 1 \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

from which we get x_1 = -2y_1 . That us , any vector x_1 = $[x_1, y_1]$ where $x_1 = -2y_1$ is an eigenvector of A with eigenvelue 1.

Plugging λ_2 into (1), we get.

$$= \begin{bmatrix} 1 & 0 \\ 1 & 3 \end{bmatrix} \quad \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \quad 3 \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}$$

from which we get $x_2 = 0$ and $y_2 \in R$. That is vector $y = [x_2, y_2]$ where $x_2 = 0$ is an eigenvector of A with eigenvalue 3.

Why is eigendecomposition is useful?

$$A[v_1 v_2] = \begin{bmatrix} 1 & 0 \\ 1 & 3 \end{bmatrix} \quad \begin{bmatrix} X_1 & Y_1 \\ Y_1 & Y_2 \end{bmatrix} \quad = \begin{bmatrix} X_1 & X_2 \\ Y_1 & Y_2 \end{bmatrix} \quad \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \quad = [v_1 v_2] \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

$V = [v_1 v_2]$

$AV = V\Lambda$

$A = V\Lambda V^{-1}$ ---------- (4)

Eigendecomposition decomposes a matrix A into a multiplication of a matrix of a eigenvectors V and a diagonal matrix of eigenvalues Λ. This can be done of a mtrix is diagonalizable. In fact, the direction of a diagonal of of a diagonalizable matrix $AR^{n\times n}$ is that it can be eigendecomposed into n eigenvectors. so that $V^{-1}AV = \Lambda$.

Matrix inverse with eigendecomposition

from (4)

$$A^{-1} = V\Lambda^{-1}V^{-1}$$

The inverse of Λ is just the inverse of each diagonal element (the eigenvalues).

Power of a matrix with eigendecomposition

from(4):

$$A^2 = V\Lambda V^{-1}V\Lambda V^{-1} = V\Lambda^2 V^{-1}$$
$$A^n = V\Lambda^n V^{-1}$$

The power of Λ is just the power of each diagonal element. This becomes much simpler than multiplications of A.

d) Singular-value decomposition

Single Value Decomposition simply known as SVD is the factorization of real or complex matrix that generalizes the eigendecomposition of a square normal matrix to any matrix via an extension of polar decomposition whereas the polar decomposition of a square real or complex matrix is any general deformation that can be uniquely decomposed into a rotation followed by a scratch component, or the stretch component followed by a rotation.

$$F = RU = vR$$

Where F is the deformation gradient,

R is the rotation tensor, which is orthogonal, $(R^{-1} = R^{\top})$ and volume conserving $(detR = 1)$,

U is the right stretch tensor, which is positive definite and symmetric $(U = U^{\top})$,

v is the left stretch tensor, which is positive definite and symmetric $(v = v^{\top})$.

If a matrix A has a matrix of eigenvalues P that is not invertible,

For example, $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$ has the non-invertible system of eigenvectors $\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$, then A does not have an eigen decomposition. However, if A is an $m \times n$ real matrix with $m > n$, then A can be written using a so called singular value decomposition of the form.

$A = UDV^T$

Note that there are several conflicting notational conventions in use in the literature where U to be an $m \times n$ matrix,

D as $n \times n$,

V as $n \times n$. However,

the Wolfram Language defines U as an $m \times m$,

D as $m \times n$, and V as $n \times n$.

In both systems, U and V have orthogonal columns so that

$U^T U = I$

and

$V^T V = I$

Note: The tow identity matrixes may have different dimensions and D has entries only along the diagonal.

$A = UDV^H$

where U and V are unitary matrix,

V^H is the conjugate transpose of V,

D is a diagonal matrix whose elements are the singular values of the original matrix.

If A is a complex matrix, then there always exists such a decomposition with positive singular values.

Singular value decomposition is implemented in the Wolfram Language as Singular Value Decomposition which retruns a $list[U, D, V]$, where U and V are matrices and D is a diagonal matrix made up of the singular values of m.

e) EVD vs SVD

From the short notes c and d, some of the differences of EVD and SVD are listed below:

1. THe EVD uses two different bases, i.e. the eigenvectors. while the SVD uses two different bases, the left and right singular vectors.
2. The basis of the eigendecomposition is not necesssarily orthogonal, the eigenbasis of the SVD is orthogonal
3. Every matrix has a SVD, it does not need to be square ot fullfill any other requirements. On the other hand, not even every square matrix has an eigendecomposition, that is a fundamental difference that makes the SVD very powerful.

f) Orthogonal Projection

Orthogonal Projection is the means of representing 3D objects 2D which is the form of parallel projection, in which all the projection lines are orthogonal to the projection plane, resulting in every plane plane of the scene appearing in affine transformation on the viewing surface. In another word, orthogonal projection is the representation of an object in which the projecting lines are at right angles to the plane of the projection.

The simple description of orthogonal projection is to remove one coordinate i.e. last coordinate.

$For\ example\ (4, 5, 9) \rightarrow (4, 5)$

An orthographic representation is rendered in the sense of geography by depicting the globe in three-dimensional space, and then eliminating the z-coordinate. While removing the z-coordinate a rotation will be applied first. The effect is to create a map that resembles the globe “as viewed from infinitely far away”. In this sense, the viewpoint projection is a limiting case, when you glance at the sky, you obtain approximately an orthographic surface image. So, for scientific illustration, the same definition (and the same terminology) is used.. Three specific orthogonal views are typically used to make three different drawings. An “Isometric” projection is a special case where a rotation is first applied, so that in the final projection the coordinate axes form angles of 120 degree with each other. The result resembles a perspective view, but with the rear parts looking slightly enlarged. A “telecentric lens” is a lens in photography which allows an orthographic projection on the focal plane.

g) PCA vs LDA

1. PCA works with the data in its entirety for the study of the key components without taking special consideration to the underlying class structure whereas LDA is specifically associated with deals discrimination between classes.
2. PCA is an unsupervised learning technique whereas LDA is a supervised learning technique that focued on class labels.
3. PCA searches for the describes the largest variations whereas LDA maximizes the correlation of the class variance and inside class variation.
4. The path of greatest variance in PCA is not generally the path of greatest discrimination, because there is no attempt to use class details, such as within class scatters and inside class scatters wheeas LDA is assumed to determine the best path of discrimination since individual densities are Gaussian with the same covariance function for both groups.
5. PCA is less superior to LDA whereas LDA is more superior to PCA
6. PCA needs less computations while LDA demands considerably more computations than PCA for large datasets.
7. The use of the PCA in the influential area of criminal enforcement is useful, whereas the LDA for diagnosis is related to issues with the detection of speech recognition.

Implementation of PCA

Some Basic Theory

Orthogonal Projections

projection of a vector x onto a 1-dimensional subspace U with basis vector b we have

$$\pi_U(x) = \frac{bb^T}{\|b\|^2}x$$

And for the general projection onto an M-dimensional subspace U with basis vectors $b_1, \ldots, b_M$ we have

$$\pi_U(x) = B(B^TB)^{-1}B^Tx$$

where

$$\mathcal{B} = [b_1, \ldots, b_M]$$

In the below code, a vector x and unit vector b where x is non-zero and not collinear with b which returns an orthogonal matrix which maps x into the line b.

In [1]:
```
def project_1d(x, b):
    assert x.ndim == 1
    assert np.allclose(1, np.sum(b**2))

    u = x - np.sign(x[0]) * np.linalg.norm(x) * b
    v = u / np.linalg.norm(u)
    p = np.eye(len(x)) - 2 * np.outer(v, v)

    return p
```

In [2]:
```
def projection_matrix_general(B):

    n, m = B.shape
    assert n >= m

    Q = np.eye(n)
    R = B.copy()

    for i in range(m - int(n==m)):
        r = R[i:, i]

        if np.allclose(r[1:], 0):
            continue
        b = np.zeros(n-i)
        b[0] = 1

        H = np.eye(n)
        H[i:, i:] = project_1d(r, b)

        Q = Q @ H.T
        R = H @ R

    return Q, R
```

Here we are using QR algorithms which is iterative at each step, B_{K+1} by taking the QR decomposition of B_K, reversing the order of Q and R, and multiply the matrices together. We are doing this for each time because off diagonals get smaller.

In [3]:
```
def eig(B, max_iter=100):
    B_k = B
    Q_k = np.eye( B.shape[1] )
    for k in range(max_iter):
        Q, R = projection_matrix_general(B_k)
        Q_k = Q_k @ Q
        B_k = R @ Q
    eigenvalues = np.diag(B_k)
    eigenvectors = Q_k
    return eigenvalues, eigenvectors
```

Assume that we are performing PCA on dataset. We then need to perform the following steps, which we break into parts:

1. Data normalization (`normalize`).
2. Find eigenvalues and corresponding eigenvectors for the covariance matrix Sort by the largest eigenvalues and the corresponding eigenvectors (`eig`).

After these steps, we can then compute the projection and reconstruction of the data onto the spaced spanned by the eigenvectors.

In [4]:
```
class PCA:
    def __init__(self, num_components=None, whiten=False):
        self.num_components = num_components
        self.whiten = bool(whiten)

    def fit(self, X):
        n, m = X.shape

        # subtract off the mean to center the data.
        self.mu = X.mean(axis=0)
        X = X - self.mu

        # whiten if necessary
        if self.whiten:
            self.std = X.std(axis=0)
            X = X / self.std

        # Eigen Decomposition of the covariance matrix
        C = X.T @ X / (n-1)
        self.eigenvalues, self.eigenvectors = projection_matrix_general(C)

        # truncate the number of components if doing dimensionality reduction
        if self.num_components is not None:
            self.eigenvalues = self.eigenvalues[0:self.num_components]
            self.eigenvectors = self.eigenvectors[:, 0:self.num_components]

        # the QR algorithm tends to puts eigenvalues in descending order
        # but is not guarenteed to. To make sure, we use argsort.
        descending_order = np.flip(np.argsort(self.eigenvalues))
        self.eigenvalues = self.eigenvalues[descending_order]
        self.eigenvectors = self.eigenvectors[:, descending_order]

        return self

    def normalize(self, X):
        X = X - self.mu

        if self.whiten:
            X = X / self.std

        return X @ self.eigenvectors

    @property
    def proportion_variance_explained(self):
        return self.eigenvalues / np.sum(self.eigenvalues)
```

Why normalization is necessary in PCA?

$$\bar{X} = \frac{X - \mu}{\sigma}$$

Ans: The term normalization is used in many contexts, with distinct, but related, meanings. Basically, normalizing means transforming so as to render normal.When data are seen as vectors, normalizing means transforming the vector so that it has unit norm. When data are though of as random variables, normalizing means transforming to normal distribution. When the data are hypothesized to be normal normalizing means transforming to unit variance.

The main reason why normalization is necessaary in PCA is that the PCA calculates a new projection of dataset. So, the new axis is centered on the variables as standard variation. And, the variable with a high standard deviation will have a higher weight for calculation of axis. So, a variable with a high standard

deviation will have a higher weight than a variable with a low standard deviation for axis calculations. If the dara is normalize, then all the variables will have the same standard deviation , thus both factors have the same weight, and the related dimension is determined by the PCA.

Now, Lets use this idea using numpy 😊

Warning: Do not import other libraries ....!

Step 1: Load the data & required libraries

We are using iris dataset which consists of three different types of irises they are Setosa, Versicolour, and Verginica which is petal and sepal length, stored in a 150 * 5 numpy.ndarray.

The rows being the samples and the columns beign: Sepal length, Sepal Width, Petal Length and Petal Width. These measures were used to create a linear discriminant model to classify the species. The dataset is often used in data mining, classification and clustering examples and to test algorithms.

In [5]:
```
%matplotlib inline
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

In [6]:
```
plt.style.use("ggplot")
plt.rcParams["figure.figsize"] = (12, 8)
```

A sample of iris data in the dataset

In [8]:
```
iris = pd.read_csv("./iris.data",
                   header = None)
iris.head()
```

Out[8]:

	0	1	2	3	4
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

In [9]:
```
iris.shape
```

Out[9]:
```
(150, 5)
```

attribute to return the column labels of the given Dataframe

In [10]:
```
iris.columns = ["sepal_length", "sepal_width", "petal_length", "petal_width", "target_cl
iris.dropna(how ='all', inplace = True)
iris.head()
```

Out[10]:

	sepal_length	sepal_width	petal_length	petal_width	target_class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

To get a concise summary of the dataframe

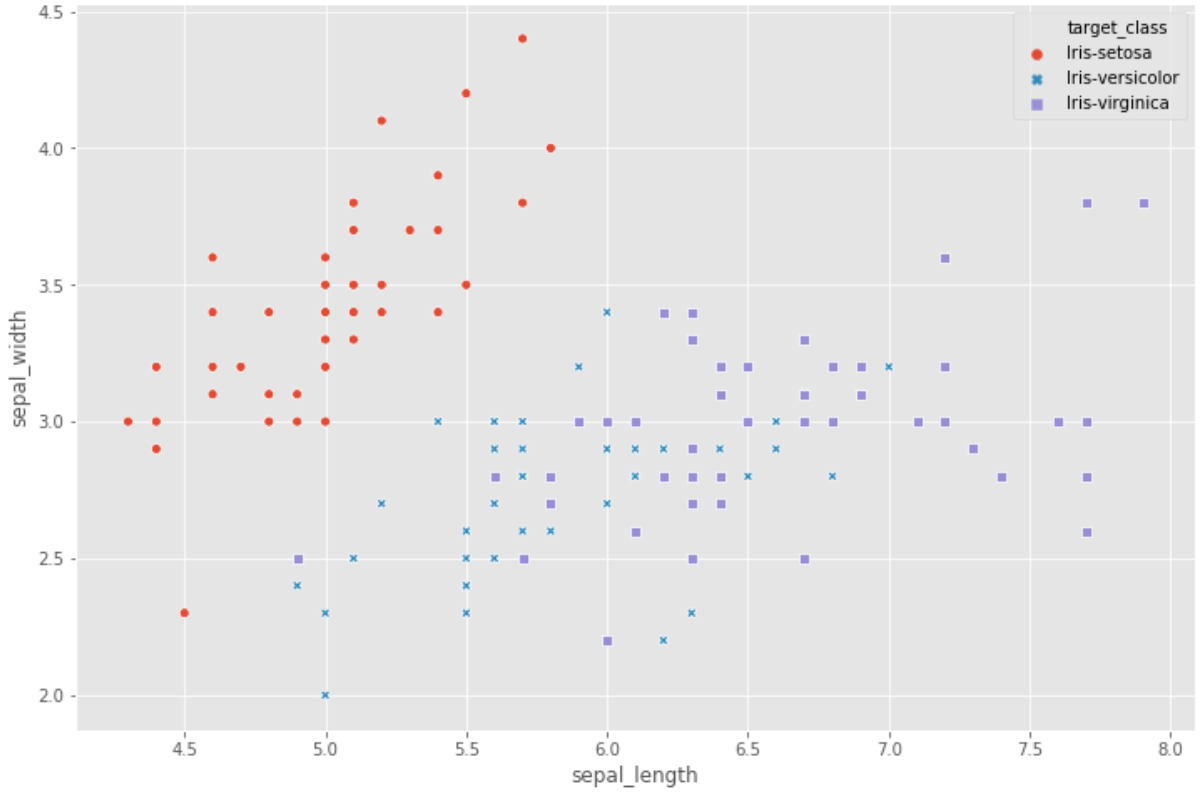
In [11]:
```
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 150 entries, 0 to 149
Data columns (total 5 columns):
sepal_length    150 non-null float64
sepal_width     150 non-null float64
petal_length    150 non-null float64
petal_width     150 non-null float64
target_class    150 non-null object
dtypes: float64(4), object(1)
memory usage: 7.0+ KB
```

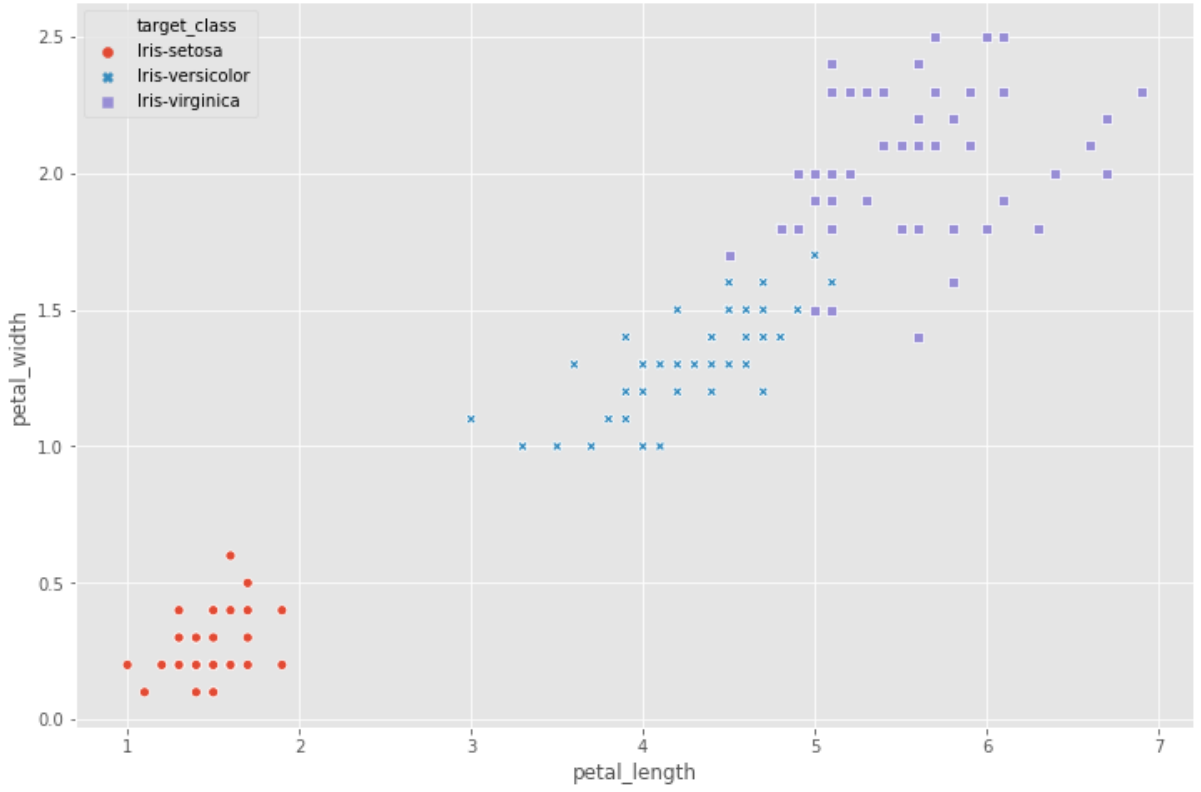
Step 2: Data Visualization

In [12]:
```
sns.scatterplot(x = iris.sepal_length, y=iris.sepal_width,hue = iris.target_class,
                style = iris.target_class)
```

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x255b8e2a588>



In [13]:
```
sns.scatterplot(x = iris.petal_length, y = iris.petal_width,
                hue = iris.target_class, style = iris.target_class);
```



Step 3: Data Pre-Processing

In [14]:
```
from sklearn.preprocessing import StandardScaler
X = iris.iloc[:, 0:4].values
y = iris.target_class.values

X = StandardScaler().fit_transform(X)
```

Step 4: Computaion of Eigen Values & Eigen Vectors

Covariance: $\sigma_{jk} = \frac{1}{n-1} \sum_{i=1}^{N} (x_{ij} - \bar{x_j})(x_{ik} - \bar{x_k})$

Coviance matrix: $\Sigma = \frac{1}{n-1}((X - \bar{x})^T (X - \bar{x}))$

In [15]:
```
covariance_matrix = np.cov(X.T)
print("Covariance matrix:\n", covariance_matrix)
```

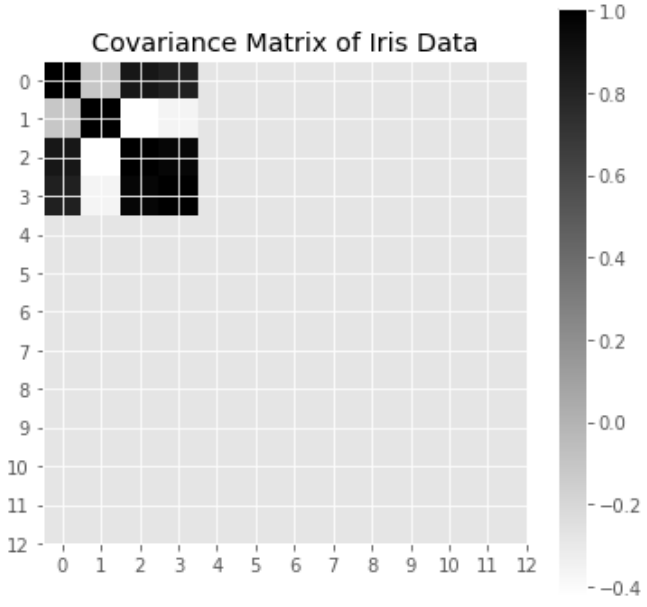
```
Covariance matrix:
 [[ 1.00671141 -0.11010327  0.87760486  0.82344326]
 [-0.11010327  1.00671141 -0.42333835 -0.358937  ]
 [ 0.87760486 -0.42333835  1.00671141  0.96921855]
 [ 0.82344326 -0.358937    0.96921855  1.00671141]]
```

Checking each features has unit variance before applying PCA

Here, Just for the purposes of this visulization, we will manually see the characteristics if they are in same scale or not.

In [16]:
```
X_data = (X - X.mean(axis=0))/X.std(axis=0)
C = X_data.T @ X_data / (X_data.shape[0] - 1)
plt.figure(figsize=(6,6))
plt.imshow(C, cmap='binary')
plt.title("Covariance Matrix of Iris Data")
plt.xticks(np.arange(0, 13, 1))
plt.yticks(np.arange(0, 13, 1))
plt.colorbar()
```

Out[16]: <matplotlib.colorbar.Colorbar at 0x255baab86a0>



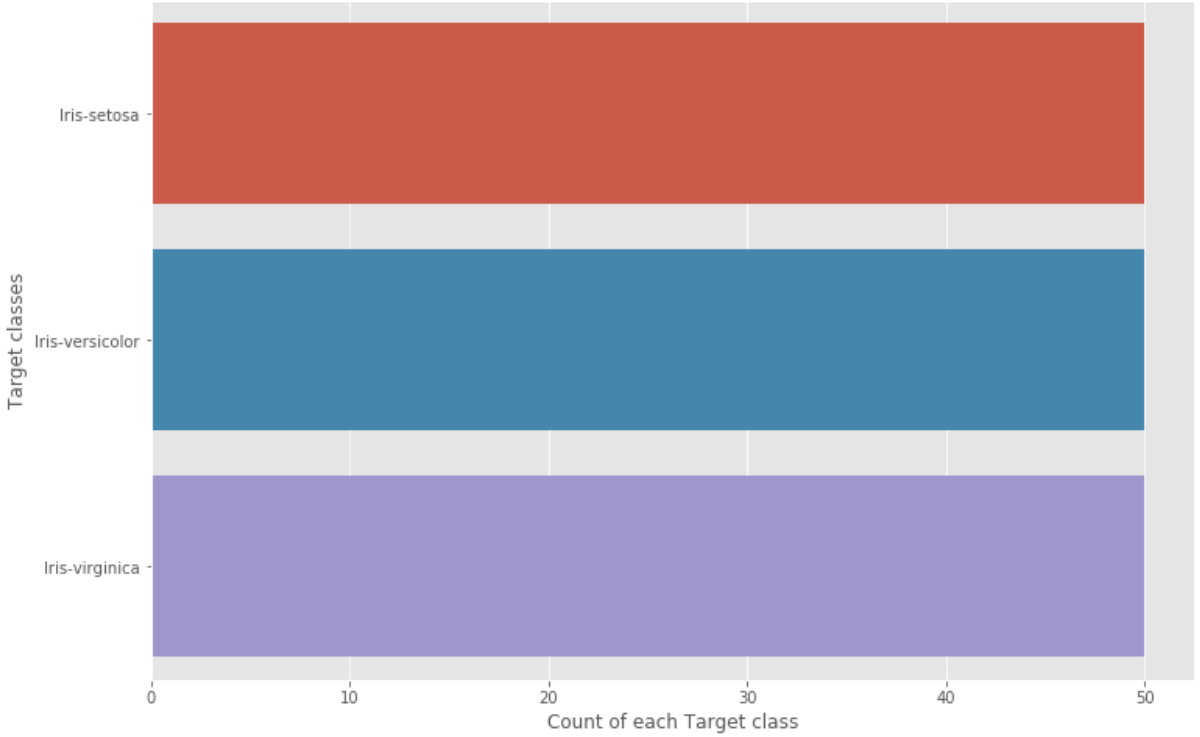
We can prove it by looking at the covariance matrix. It has the property that it is symmetric. We also constrain the each of the columns (eigenvectors) such that the values sum to 1. Thus, they are orthonormal to each other.

Eigen decomposition of covariance matrix : $\Sigma = W \wedge W^{-1}$

W is the matrix composed by total eigenvectors, $\wedge$ is the diagonal matrix with all eigenvalues in diagonal.

IT is clear from this plot that above dataset exhibits significant multicollinearity. Every feature exhibiting high correlations with several others; no feature is truly independent. While that would be a bad thing for say, linear regression, it means these data are an ideal candidate for PCA.

In [17]:
```
sns.countplot(y=iris.target_class ,data=iris)
plt.xlabel("Count of each Target class")
plt.ylabel("Target classes")
plt.show()
```



In [18]:
```
#calculate eigen_values, eigen_vectors using numpy
eigen_values, eigen_vectors = np.linalg.eig(covariance_matrix)
print("Eigenvectors:\n", eigen_vectors, "\n")
print("Eigenvalues:\n", eigen_values)
```

```
Eigenvectors:
 [[ 0.52237162 -0.37231836 -0.72101681  0.26199559]
 [-0.26335492 -0.92555649  0.24203288 -0.12413481]
 [ 0.58125401 -0.02109478  0.14089226 -0.80115427]
 [ 0.56561105 -0.06541577  0.6338014   0.52354627]]

Eigenvalues:
 [2.93035378 0.92740362 0.14834223 0.02074601]
```

Step 5: Singular Value Decomposition (SVD)

This is basically used to diagonalize the feature matrix when it is not a square matrix.

In [19]:
```
evec_svd, s, v = np.linalg.svd(X.T)
print("Eigenvectors \n", evec_svd, "\n")

#here, we just interested in directions of vector, so sign doesn't matter
```

```
Eigenvectors
 [[-0.52237162 -0.37231836  0.72101681  0.26199559]
 [ 0.26335492 -0.92555649 -0.24203288 -0.12413481]
 [-0.58125401 -0.02109478 -0.14089226 -0.80115427]
 [-0.56561105 -0.06541577 -0.6338014   0.52354627]]
```

What is major difference between EVD & SVD?

Consider Eigen Value Decompositio $A = PDP^{-1}$ and
Single Value Decomposition SVD $(A) = U\Sigma V^*$

The major difference of EVD and SVD is that in the SVD the entries in the diagonal matrix Σ are all real and nonnegative. In the eigendecomposition, the entries of D can be any complex number - negative, positive, imaginary, whatever.

Step 6: Picking Principal Components Using the Explained Variance

In [20]:
```
for val in eigen_values:
    print(val)
```

```
2.9303537755893165
0.9274036215173417
0.1483422264816399
0.02074601399559571
```

In [21]:
```
eigen_vec_svd, _, _= np.linalg.svd(X.T)
eigen_vec_svd
```

Out[21]:
```
array([[-0.52237162, -0.37231836,  0.72101681,  0.26199559],
       [ 0.26335492, -0.92555649, -0.24203288, -0.12413481],
       [-0.58125401, -0.02109478, -0.14089226, -0.80115427],
       [-0.56561105, -0.06541577, -0.6338014 ,  0.52354627]])
```

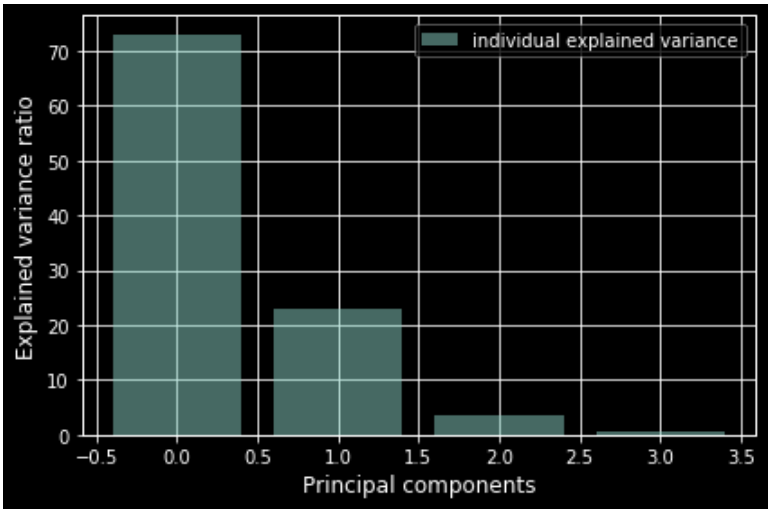
In [22]:
```
variance_explained = [(i/sum(eigen_values))*100 for i in eigen_values]
print("Variance Explained: \n", variance_explained)

#72% variance in our data is explained by the first eigen-value i.e, 2.930...
```

```
Variance Explained:
 [72.77045209380134, 23.030523267680643, 3.683831957627389, 0.5151926808906266]
```

In [23]:
```
with plt.style.context('dark_background'):
    plt.figure(figsize=(6, 4))

    plt.bar(range(4), variance_explained, alpha=0.5, align='center',
            label='individual explained variance')
    plt.ylabel('Explained variance ratio')
    plt.xlabel('Principal components')
    plt.legend(loc='best')
    plt.tight_layout()
```



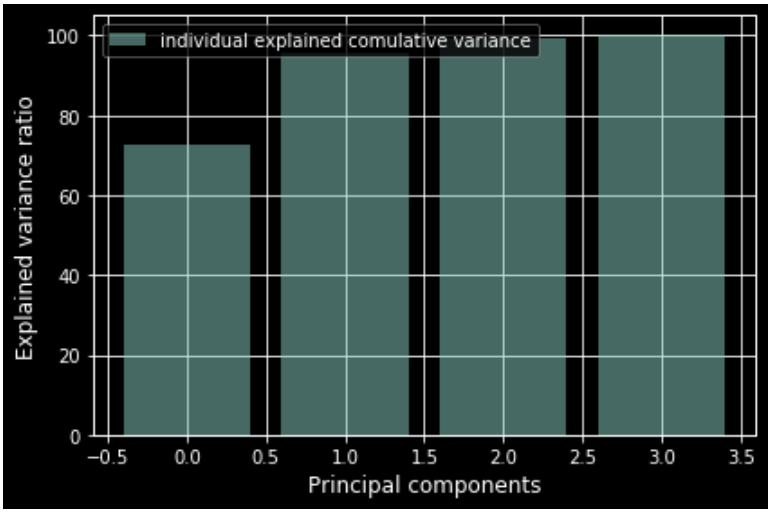
In [24]:
```
cumulative_variance_explained = np.cumsum(variance_explained)
print(cumulative_variance_explained)

#If we used only 1st eigen value to choose principal components, We will be able to capt
# If we choose 2 eigen values, 95% is handled and so on...
```

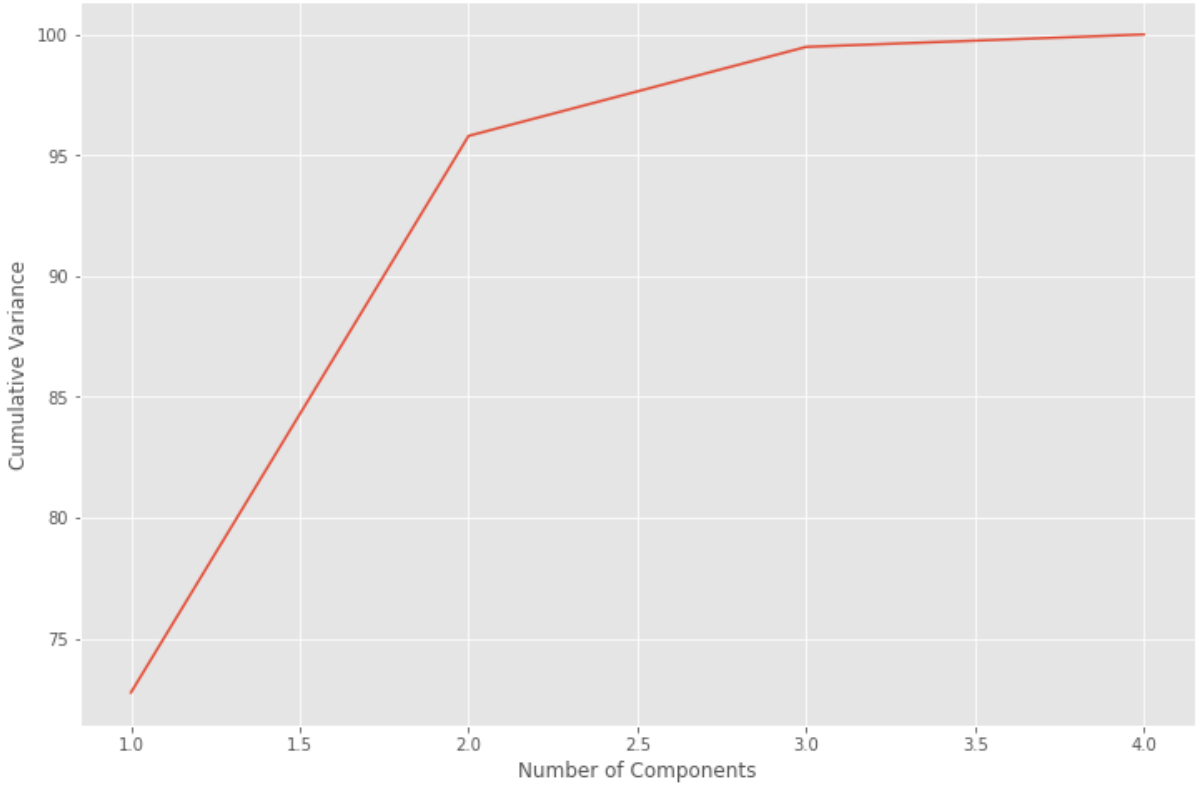
```
[ 72.77045209  95.80097536  99.48480732 100.        ]
```

In [25]:
```
with plt.style.context('dark_background'):
    plt.figure(figsize=(6, 4))

    plt.bar(range(4), cumulative_variance_explained, alpha=0.5, align='center',
            label='individual explained comulative variance')
    plt.ylabel('Explained variance ratio')
    plt.xlabel('Principal components')
    plt.legend(loc='best')
    plt.tight_layout()
```



In [26]:
```
sns.lineplot(x = [1, 2, 3, 4], y = cumulative_variance_explained)
plt.xlabel ('Number of Components')
plt.ylabel('Cumulative Variance')
plt.show()
```



Step 7: Project Data onto Lower Dimensional Linear Subspace

In [27]:
```
for val in eigen_vectors:
    print(val)
```

```
[ 0.52237162 -0.37231836 -0.72101681  0.26199559]
[-0.26335492 -0.92555649  0.24203288 -0.12413481]
[ 0.58125401 -0.02109478  0.14089226 -0.80115427]
[ 0.56561105 -0.06541577  0.6338014   0.52354627]
```

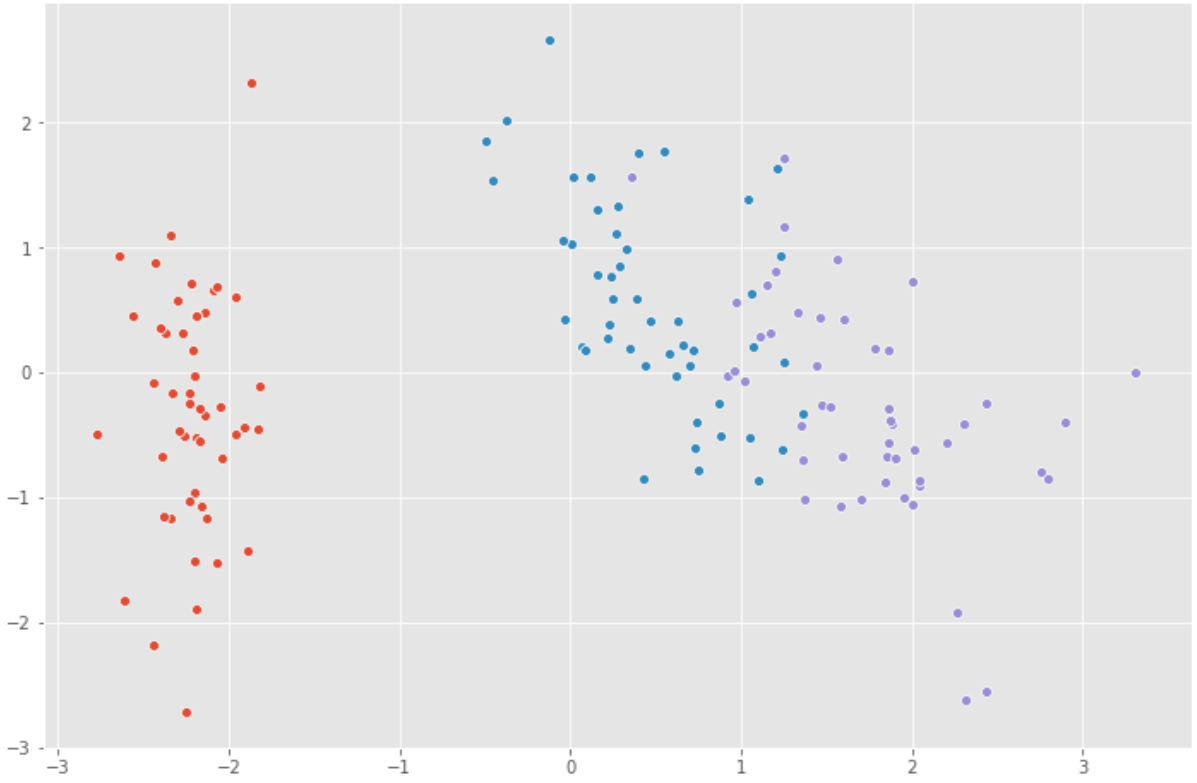
In [28]:
```
aa = eigen_vectors.T[:][:]
bb = aa[:2]
projection_matrix = bb.T
projection_matrix
```

Out[28]:
```
array([[ 0.52237162, -0.37231836],
       [-0.26335492, -0.92555649],
       [ 0.58125401, -0.02109478],
       [ 0.56561105, -0.06541577]])
```

In [29]:
```
x_pca = X.dot(projection_matrix)
x_pca
```

Out[29]:
```
array([[-2.26454173e+00, -5.05703903e-01],
       [-2.08642550e+00,  6.55404729e-01],
       [-2.36795045e+00,  3.18477311e-01],
       [-2.30419716e+00,  5.75367713e-01],
       [-2.38877749e+00, -6.74767397e-01],
       [-2.07053681e+00, -1.51854856e+00],
       [-2.44571134e+00, -7.45626750e-02],
       [-2.23384186e+00, -2.47613932e-01],
       [-2.34195768e+00,  1.09514636e+00],
       [-2.18867576e+00,  4.48629048e-01],
       [-2.16348656e+00, -1.07059558e+00],
       [-2.32737775e+00, -1.58587455e-01],
       [-2.22408272e+00,  7.09118158e-01],
       [-2.63971626e+00,  9.38281982e-01],
       [-2.19229151e+00, -1.88997851e+00],
       [-2.25146521e+00, -2.72237108e+00],
       [-2.20275048e+00, -1.51375028e+00],
       [-2.19017916e+00, -5.14304308e-01],
       [-1.89407429e+00, -1.43111071e+00],
```

In [30]:
```
for target_class in ('Iris-setosa', 'Iris-versicolor', 'Iris-virginica'):
    sns.scatterplot (x_pca[y == target_class, 0],
                     x_pca[y == target_class, 1])
```



Can we classify after dimensionality reduction?

https://towardsdatascience.com/dimensionality-reduction-does-pca-really-improve-classification-outcome-6e9ba21f0a32 (https://towardsdatascience.com/dimensionality-reduction-does-pca-really-improve-classification-outcome-6e9ba21f0a32)

In the modern scientific era, increasing quantities of data are being produced and collected. How, in

machine learning, too much data can be a bad thing. At a certain level, additional features or dimensions will decrease the precision of a model, because more data has to be generalized. Thus, this is recognized as the "Curse of dimensionality".

Dimensionality Reduction is a way to reduce a model's difficulty and prevent overfitting. Feature Selection and Feature Extraction are two main types of dimensionality reduction. We choose a subset of the original features by selecting a feature, while in feature extraction we draw knowledge from the component set to create a new feature subspace. Extraction of features is not only used to improve storage space or the learning algorithm's computational efficiency, but can also improve predictive performance by reducing the dimensionality curse — especially if we are working with non-regularized models. Also Linear Discriminant Analysis (LDA) and Principal Component Analysis (PCA) are the two key dimensional reduction algorithms.

Classification of high-dimensional data, such as photographs, data on gene expression and spectral data, presents an important challenge to machine learning as predictive models based on these data run the risk of over-fitting or the existence of a large number of redundant or strongly associated attributes will significantly degrade classification accuracy. The goal is therefore to use PCA algorithms to the high dimensional data and to boost the predictive efficiency of several well-known machine learning algorithms. In other words, PCA is a classic computational approach for converting dataset attributes into a new collection of uncorrelated attributes called Principal Components, which increases the efficiency of machine learning when processing high-dimensional data. In simple terms, what we can say is that, PCA finds a new, lower dimensional orthonormal base such that the largest variance of the original data is kept. In addition, these low errors have been achieved through PCA, despite a significant reduction in data from the original attributes to at least the minimum attributes. In addition, more than 20 PCs are expected to find the optimum data set because the efficiency of the majority classifiers is decreasing with a growing number of PCs.

The PCA was particularly suited to the proposed approach, since it does not require the generation of all PCs with a data matrix, in contrast to the widely used proprietary decomposition methods and the first-derived pre-processing technique followed by standardization, it improves the performance of the majority of the classification tasks in machine learning.

Question may arise: what if we used data mining techniques for large dataset without PCA?

This is because smaller datasets are much easier to visualize and explore and analyze data for machine learning algorithms without extraneous variables to process. The premises underlying the PCA are causal, so the definition is only accurate if the assumptions are correct. PCA computation on nonlinear data or broad dataset would then have little significance, only decomposing to the dominant linear nodes a global linear representation of the distribution of data is given.