**Cross Validation in Machine Learning**

In **machine learning**, we couldn’t fit the model on the training data and can’t say that the model will work accurately for the real data. For this, we must assure that our model got the correct patterns from the data, and it is not getting up too much noise. For this purpose, we use the **cross-validation technique**. In this article, we’ll delve into the process of cross-validation in machine learning.

**What is Cross-Validation?**

Cross validation is a technique used in machine learning to evaluate the performance of a model on unseen data. It involves dividing the available data into multiple folds or subsets, using one of these folds as a validation set, and training the model on the remaining folds. This process is repeated multiple times, each time using a different fold as the validation set. Finally, the results from each validation step are averaged to produce a more robust estimate of the model’s performance. Cross validation is an important step in the [machine learning](https://www.geeksforgeeks.org/machine-learning/) process and helps to ensure that the model selected for deployment is robust and generalizes well to new data.

**What is cross-validation used for?**

The main purpose of cross validation is to prevent [overfitting](https://www.geeksforgeeks.org/overfitting-and-regularization-in-ml/), which occurs when a model is trained too well on the training data and performs poorly on new, unseen data. By evaluating the model on multiple validation sets, cross validation provides a more realistic estimate of the model’s generalization performance, i.e., its ability to perform well on new, unseen data.

**Types of Cross-Validation**

There are several types of cross validation techniques, including **k-fold cross validation, leave-one-out cross validation, and Holdout validation, Stratified Cross-Validation.**The choice of technique depends on the size and nature of the data, as well as the specific requirements of the modeling problem.

**1. Holdout Validation**

In[Holdout Validation](https://www.geeksforgeeks.org/introduction-of-holdout-method/), we perform training on the 50% of the given dataset and rest 50% is used for the testing purpose. It’s a simple and quick way to evaluate a model. The major drawback of this method is that we perform training on the 50% of the dataset, it may possible that the remaining 50% of the data contains some important information which we are leaving while training our model i.e. higher bias.

**2. LOOCV (Leave One Out Cross Validation)**

In this method, we perform training on the whole dataset but leaves only one data-point of the available dataset and then iterates for each data-point. In [LOOCV](https://www.geeksforgeeks.org/loocvleave-one-out-cross-validation-in-r-programming/), the model is trained on  samples and tested on the one omitted sample, repeating this process for each data point in the dataset. It has some advantages as well as disadvantages also.

**An advantage** of using this method is that we make use of all data points and hence it is low bias.

The major**drawback**of this method is that it leads to **higher variation**in the testing model as we are testing against one data point. If the data point is an outlier it can lead to higher variation. Another drawback is it **takes a lot of execution time** as it iterates over ‘the number of data points’ times.

**3. Stratified Cross-Validation**

It is a technique used in machine learning to ensure that each fold of the cross-validation process maintains the same class distribution as the entire dataset. This is particularly important when dealing with imbalanced datasets, where certain classes may be underrepresented. In this method,

1. The dataset is divided into k folds while maintaining the proportion of classes in each fold.
2. During each iteration, one-fold is used for testing, and the remaining folds are used for training.
3. The process is repeated k times, with each fold serving as the test set exactly once.

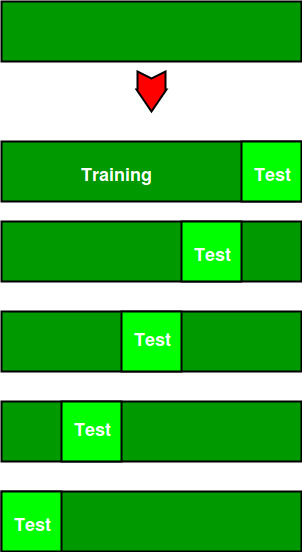
[Stratified Cross-Validation](https://www.geeksforgeeks.org/stratified-k-fold-cross-validation/)is essential when dealing with classification problems where maintaining the balance of class distribution is crucial for the model to generalize well to unseen data.

**4. K-Fold Cross Validation**

In [K-Fold Cross Validation](https://www.geeksforgeeks.org/k-fold-cross-validation-in-r-programming/), we split the dataset into k number of subsets (known as folds) then we perform training on the all the subsets but leave one(k-1) subset for the evaluation of the trained model. In this method, we iterate k times with a different subset reserved for testing purpose each time.

***Note:****It is always suggested that the value of k should be 10 as the lower value of k is takes towards validation and higher value of k leads to LOOCV method.*

**Example of K Fold Cross Validation**

The diagram below shows an example of the training subsets and evaluation subsets generated in k-fold cross-validation. Here, we have total 25 instances. In first iteration we use the first 20 percent of data for evaluation, and the remaining 80 percent for training ([1-5] testing and [5-25] training) while in the second iteration we use the second subset of 20 percent for evaluation, and the remaining three subsets of the data for training ([5-10] testing and [1-5 and 10-25] training), and so on.

Total instances: 25  
Value of k : 5   
No. Iteration Training set observations Testing set observations  
 1 [ 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24] [0 1 2 3 4]  
 2 [ 0 1 2 3 4 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24] [5 6 7 8 9]  
 3 [ 0 1 2 3 4 5 6 7 8 9 15 16 17 18 19 20 21 22 23 24] [10 11 12 13 14]  
 4 [ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 20 21 22 23 24] [15 16 17 18 19]  
 5 [ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19] [20 21 22 23 24]

**Comparison between cross-validation and hold out method**

**Advantages of train/test split:**

1. This runs K times faster than Leave One Out cross-validation because K-fold cross-validation repeats the train/test split K-times.
2. Simpler to examine the detailed results of the testing process.

**Advantages of cross-validation:**

1. More accurate estimate of out-of-sample accuracy.
2. More “efficient” use of data as every observation is used for both training and testing.

**Advantages and Disadvantages of Cross Validation**

**Advantages:**

1. Overcoming Overfitting: Cross validation helps to prevent overfitting by providing a more robust estimate of the model’s performance on unseen data.
2. Model Selection: Cross validation can be used to compare different models and select the one that performs the best on average.
3. Hyperparameter tuning: Cross validation can be used to optimize the hyperparameters of a model, such as the regularization parameter, by selecting the values that result in the best performance on the validation set.
4. Data Efficient: Cross validation allows the use of all the available data for both training and validation, making it a more data-efficient method compared to traditional validation techniques.

**Disadvantages:**

1. Computationally Expensive: Cross validation can be computationally expensive, especially when the number of folds is large or when the model is complex and requires a long time to train.
2. Time-Consuming: Cross validation can be time-consuming, especially when there are many hyperparameters to tune or when multiple models need to be compared.
3. Bias-Variance Tradeoff: The choice of the number of folds in cross validation can impact the bias-variance tradeoff, i.e., too few folds may result in high variance, while too many folds may result in high bias.

**Python implementation for k fold cross-validation**

**Step 1: Import necessary libraries.**

* Python3

|  |
| --- |
| **from** sklearn.model\_selection **import** cross\_val\_score, KFold  **from** sklearn.svm **import** SVC  **from** sklearn.datasets **import** load\_iris |

**Step 2: Load the dataset**

let’s use the iris dataset, a multi-class classification in-built dataset.

* Python3

|  |
| --- |
| iris **=** load\_iris()  X, y **=** iris.data, iris.target |

**Step 3: Create SVM classifier**

SVC is a Support Vector Classification model from scikit-learn.

* Python3

|  |
| --- |
| svm\_classifier **=** SVC(kernel**=**'linear') |

**Step 4:Define the number of folds for cross-validation**

* Python3

|  |
| --- |
| num\_folds **=** 5  kf **=** KFold(n\_splits**=**num\_folds, shuffle**=**True, random\_state**=**42) |

**Step 5: Perform k-fold cross-validation**

* Python3

|  |
| --- |
| cross\_val\_results **=** cross\_val\_score(svm\_classifier, X, y, cv**=**kf) |

**Step 6: Evaluation metrics**

* Python3

|  |
| --- |
| print(f'Cross-Validation Results (Accuracy): {cross\_val\_results}')  print(f'Mean Accuracy: {cross\_val\_results.mean()}') |

**Output:**

Cross-Validation Results (Accuracy): [1. 1. 0.96666667 0.93333333 0.96666667]  
Mean Accuracy: 0.9733333333333334

**Frequently Asked Questions(FAQs)**

**1.What is K in K fold cross validation?**

*It represents the number of folds or subsets into which the dataset is divided for cross-validation. Common values are 5 or 10.*

**2.How many folds for cross-validation?**

*The number of folds is a parameter in K-fold cross-validation, typically set to 5 or 10. It determines how many subsets the dataset is divided into.*

**3.What is cross-validation example?**

*Split the dataset into five folds. For each fold, train the model on four folds and evaluate it on the remaining fold. The average performance across all five folds is the estimated out-of-sample accuracy.*

**4.What is the purpose of validation?**

*Validation assesses a model’s performance on unseen data, helping detect overfitting. It ensures the model generalizes well and is not just memorizing the training data.*

**5. Why use 10-fold cross-validation?**

*10-fold cross-validation provides a balance between robust evaluation and computational efficiency. It offers a good trade-off by dividing the data into 10 subsets for comprehensive assessment.*

**Bagging vs Boosting in Machine Learning**

As we know, [Ensemble learning](https://www.geeksforgeeks.org/ensemble-classifier-data-mining/) helps improve machine learning results by combining several models. This approach allows the production of better predictive performance compared to a single model. Basic idea is to learn a set of classifiers (experts) and to allow them to vote. **Bagging** and **Boosting**are two types of **Ensemble Learning**. These two decrease the variance of a single estimate as they combine several estimates from different models. So the result may be a model with higher stability. Let’s understand these two terms in a glimpse.

1. **Bagging**: It is a homogeneous weak learners’ model that learns from each other independently in parallel and combines them for determining the model average.
2. **Boosting**: It is also a homogeneous weak learners’ model but works differently from Bagging. In this model, learners learn sequentially and adaptively to improve model predictions of a learning algorithm.

Let’s look at both of them in detail and understand the Difference between Bagging and Boosting.

**Bagging**

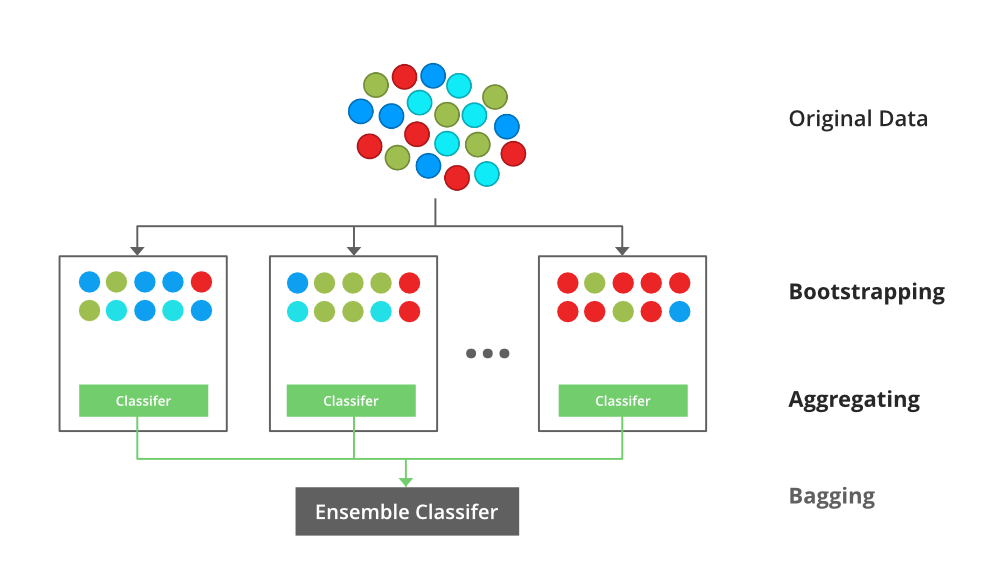
**B**ootstrap **A**ggregating, also known as bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It decreases the [variance](https://www.geeksforgeeks.org/mathematics-mean-variance-and-standard-deviation/)and helps to avoid [overfitting](https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/). It is usually applied to [decision tree methods](https://www.geeksforgeeks.org/decision-tree/). Bagging is a special case of the model averaging approach.

**Description of the Technique**

Suppose a set D of d tuples, at each iteration i, a training set Di of d tuples is selected via row sampling with a replacement method (i.e., there can be repetitive elements from different d tuples) from D (i.e., bootstrap). Then a classifier model Mi is learned for each training set D < i. Each classifier Mi returns its class prediction. The bagged classifier M\* counts the votes and assigns the class with the most votes to X (unknown sample).

**Implementation Steps of Bagging**

* **Step 1:** Multiple subsets are created from the original data set with equal tuples, selecting observations with replacement.
* **Step 2:** A base model is created on each of these subsets.
* **Step 3:**Each model is learned in parallel with each training set and independent of each other.
* **Step 4:**The final predictions are determined by combining the predictions from all the models.



*An illustration for the concept of bootstrap aggregating (Bagging)*

**Example of Bagging**

The [Random Forest model](https://www.geeksforgeeks.org/random-forest-regression-in-python/) uses Bagging, where decision tree models with higher variance are present. It makes random feature selection to grow trees. Several random trees make a Random Forest.

*To read more refer to this article:* [Bagging classifier](https://www.geeksforgeeks.org/ml-bagging-classifier/)

**Boosting**

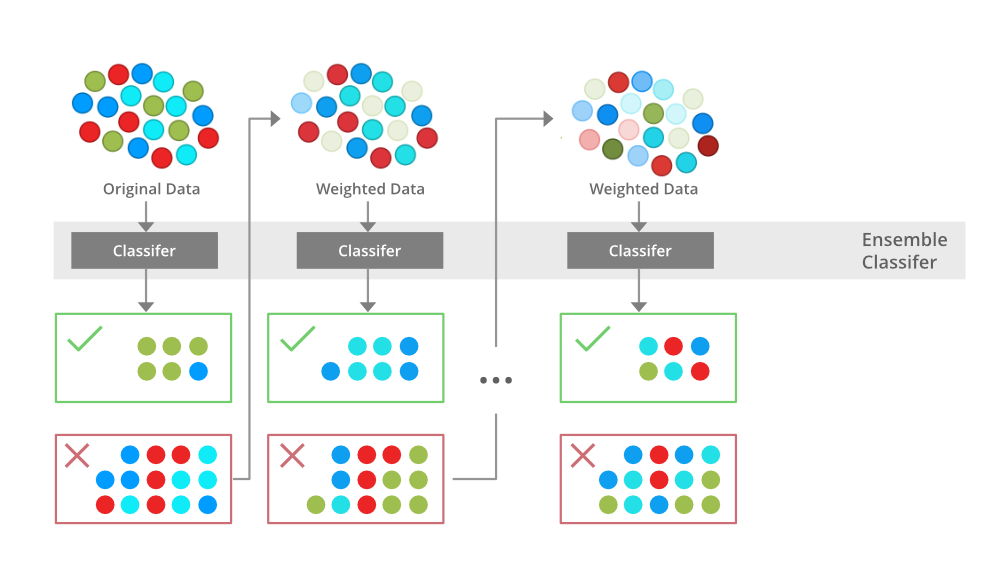
Boosting is an ensemble modeling technique that attempts to build a strong classifier from the number of weak classifiers. It is done by building a model by using weak models in series. Firstly, a model is built from the training data. Then the second model is built which tries to correct the errors present in the first model. This procedure is continued and models are added until either the complete training data set is predicted correctly or the maximum number of models is added.

**Boosting Algorithms**

There are several boosting algorithms. The original ones, proposed by **Robert Schapire** and **Yoav Freund** were not adaptive and could not take full advantage of the weak learners. Schapire and Freund then developed [AdaBoost](https://www.geeksforgeeks.org/implementing-the-adaboost-algorithm-from-scratch/), an adaptive boosting algorithm that won the prestigious Gödel Prize. AdaBoost was the first really successful boosting algorithm developed for the purpose of binary classification. AdaBoost is short for Adaptive Boosting and is a very popular boosting technique that combines multiple “weak classifiers” into a single “strong classifier”.

***Algorithm:***

1. *Initialise the dataset and assign equal weight to each of the data point.*
2. *Provide this as input to the model and identify the wrongly classified data points.*
3. *Increase the weight of the wrongly classified data points and decrease the weights of correctly classified data points. And then normalize the weights of all data points.*
4. *if (got required results)  
     Goto step 5  
   else  
     Goto step 2*
5. *End*



*An illustration presenting the intuition behind the boosting algorithm, consisting of the parallel learners and weighted dataset.*

*To read more refer to this article:*[Boosting and AdaBoost in ML](https://www.geeksforgeeks.org/boosting-in-machine-learning-boosting-and-adaboost/)

**Similarities Between Bagging and Boosting**

Bagging and Boosting, both being the commonly used methods, have a universal similarity of being classified as ensemble methods. Here we will explain the similarities between them.

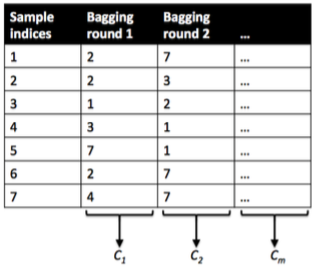
1. Both are ensemble methods to get N learners from 1 learner.
2. Both generate several training data sets by random sampling.
3. Both make the final decision by averaging the N learners (or taking the majority of them i.e Majority Voting).
4. Both are good at reducing variance and provide higher stability.

**Differences Between Bagging and Boosting**

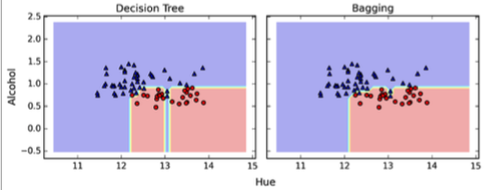
| **S.NO** | **Bagging** | **Boosting** |
| --- | --- | --- |
| **1.** | **The simplest way of combining predictions that  belong to the same type.** | **A way of combining predictions that  belong to the different types.** |
| **2.** | **Aim to decrease variance, not bias.** | **Aim to decrease bias, not variance.** |
| **3.** | **Each model receives equal weight.** | **Models are weighted according to their performance.** |
| **4.** | **Each model is built independently.** | **New models are influenced  by the performance of previously built models.** |
| **5.** | **Different training data subsets are selected using row sampling with replacement and random sampling methods from the entire training dataset.** | **Every new subset contains the elements that were misclassified by previous models.** |
| **6.** | **Bagging tries to solve the over-fitting problem.** | **Boosting tries to reduce bias.** |
| **7.** | **If the classifier is unstable (high variance), then apply bagging.** | **If the classifier is stable and simple (high bias) the apply boosting.** |
| **8.** | **In this base classifiers are trained parallelly.** | **In this base classifiers are trained sequentially.** |
| **9** | **Example: The Random forest model uses Bagging.** | **Example: The AdaBoost uses Boosting techniques** |

Bagging

Now, let’s take a look at the probably “simplest” case, bagging. Here, we train a number (ensemble) of decision trees from bootstrap samples of your training set. Bootstrap sampling means drawing random samples from our training set with replacement. E.g., if our training set consists of 7 training samples, our bootstrap samples (here: n=7) can look as follows, where C1, C2, … Cm shall symbolize the decision tree classifiers:

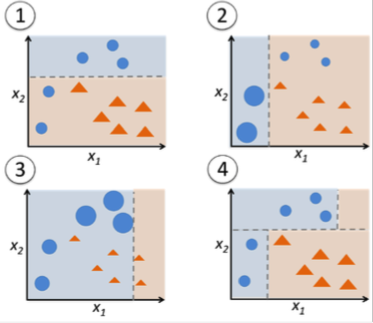


After we trained your (m) decision trees, we can use them to classify new data via majority rule. For instance, we’d let each decision tree make a decision and predict the class label that received more votes. Typically, this would result in a less complex decision boundary, and the bagging classifier would have a lower variance (less overfitting) than an individual decision tree. Below is a plot comparing a single decision tree (left) to a bagging classifier (right) for 2 variables from the Wine dataset (Alcohol and Hue).

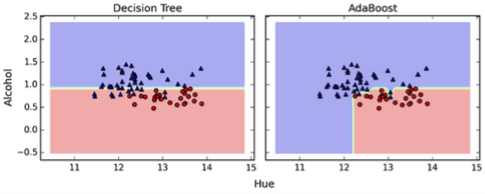


Boosting

In contrast to bagging, you use very simple classifiers as base classifiers, so-called “weak learners.” Picture these weak learners as “decision tree stumps” – decision trees with only 1 splitting rule. Below, we will refer to the probably most popular example of boosting, AdaBoost. Here, we start with one decision tree stump (1) and “focus” on the samples it got wrong. In the next round, we train another decision tree stump that attempts to get these samples right (2); we achieve this by putting a larger weight on these training samples. Again, this 2nd classifier will likely get some other samples wrong, so you’d re-adjust the weights …



In a nutshell, we can summarize “Adaboost” as “adaptive” or “incremental” learning from mistakes. Eventually, we will come up with a model that has a lower bias than an individual decision tree (thus, it is less likely to underfit the training data).



Random forests

The random forest algorithm is actually a bagging algorithm: also here, we draw random bootstrap samples from your training set. However, in addition to the bootstrap samples, we also draw random subsets of features for training the individual trees; in bagging, we provide each tree with the full set of features. Due to the random feature selection, the trees are more independent of each other compared to regular bagging, which often results in better predictive performance (due to better variance-bias trade-offs), and I’d say that it’s also faster than bagging, because each tree learns only from a subset of features.

**Principal Component Analysis(PCA)**

As the number of features or dimensions in a dataset increases, the amount of data required to obtain a statistically significant result increases exponentially. This can lead to issues such as overfitting, increased computation time, and reduced accuracy of machine learning models this is known as the curse of dimensionality problems that arise while working with high-dimensional data.

As the number of dimensions increases, the number of possible combinations of features increases exponentially, which makes it computationally difficult to obtain a representative sample of the data and it becomes expensive to perform tasks such as clustering or classification because it becomes. Additionally, some [machine learning](https://www.geeksforgeeks.org/machine-learning/) algorithms can be sensitive to the number of dimensions, requiring more data to achieve the same level of accuracy as lower-dimensional data.

To address the [curse of dimensionality](https://www.geeksforgeeks.org/videos/curse-of-dimensionality-in-machine-learning/), [Feature engineering](https://www.geeksforgeeks.org/what-is-feature-engineering/)techniques are used which include feature selection and feature extraction. [Dimensionality reduction](https://www.geeksforgeeks.org/dimensionality-reduction/) is a type of feature extraction technique that aims to reduce the number of input features while retaining as much of the original information as possible.

In this article, we will discuss one of the most popular dimensionality reduction techniques i.e. Principal Component Analysis(PCA).

**What is Principal Component Analysis(PCA)?**

[Principal Component Analysis](https://www.geeksforgeeks.org/principal-component-analysis-with-python/)(PCA) technique was introduced by the mathematician **Karl Pearson** in 1901**.** It works on the condition that while the data in a higher dimensional space is mapped to data in a lower dimension space, the variance of the data in the lower dimensional space should be maximum.

* **Principal Component Analysis (PCA)**is a statistical procedure that uses an orthogonal transformation that converts a set of correlated variables to a set of uncorrelated variables.PCA is the most widely used tool in exploratory data analysis and in machine learning for predictive models. Moreover,
* Principal Component Analysis (PCA) is an [unsupervised learning](https://www.geeksforgeeks.org/supervised-unsupervised-learning/) algorithm technique used to examine the interrelations among a set of variables. It is also known as a general factor analysis where regression determines a line of best fit.
* The main goal of Principal Component Analysis (PCA) is to reduce the dimensionality of a dataset while preserving the most important patterns or relationships between the variables without any prior knowledge of the target variables.

Principal Component Analysis (PCA) is used to reduce the dimensionality of a data set by finding a new set of variables, smaller than the original set of variables, retaining most of the sample’s information, and useful for the [regression and classification](https://www.geeksforgeeks.org/regression-classification-supervised-machine-learning/) of data.*Principal Component Analysis*

1. Principal Component Analysis (PCA) is a technique for dimensionality reduction that identifies a set of orthogonal axes, called principal components, that capture the maximum variance in the data. The principal components are linear combinations of the original variables in the dataset and are ordered in decreasing order of importance. The total variance captured by all the principal components is equal to the total variance in the original dataset.
2. The first principal component captures the most variation in the data, but the second principal component captures the maximum [variance](https://www.geeksforgeeks.org/python-statistics-variance/) that is [orthogonal](https://www.geeksforgeeks.org/orthogonal-and-orthonormal-vectors-in-linear-algebra/) to the first principal component, and so on.
3. Principal Component Analysis can be used for a variety of purposes, including data visualization, feature selection, and data compression. In data visualization, PCA can be used to plot high-dimensional data in two or three dimensions, making it easier to interpret. In feature selection, PCA can be used to identify the most important variables in a dataset. In data compression, PCA can be used to reduce the size of a dataset without losing important information.
4. In Principal Component Analysis, it is assumed that the information is carried in the variance of the features, that is, the higher the variation in a feature, the more information that features carries.

Overall, PCA is a powerful tool for data analysis and can help to simplify complex datasets, making them easier to understand and work with.

**Step-By-Step Explanation of PCA (Principal Component Analysis)**

**Step 1: Standardization**

First, we need to [standardize](https://www.geeksforgeeks.org/normalization-vs-standardization/) our dataset to ensure that each variable has a mean of 0 and a standard deviation of

1.

Here,

* is the mean of independent features
* is the [standard deviation](https://www.geeksforgeeks.org/mathematics-mean-variance-and-standard-deviation/) of independent features

**Step2: Covariance Matrix Computation**

[Covariance](https://www.geeksforgeeks.org/mathematics-covariance-and-correlation/) measures the strength of joint variability between two or more variables, indicating how much they change in relation to each other. To find the covariance we can use the formula:

The value of covariance can be positive, negative, or zeros.

* Positive: As the x1 increases x2 also increases.
* Negative: As the x1 increases x2 also decreases.
* Zeros: No direct relation

**Step 3: Compute Eigenvalues and Eigenvectors of Covariance Matrix to Identify Principal Components**

Let A be a square nXn matrix and X be a non-zero vector for which

for some scalar values . then  is known as the [eigenvalue](https://www.geeksforgeeks.org/how-to-compute-the-eigenvalues-and-right-eigenvectors-of-a-given-square-array-using-numpy/) of matrix A and X is known as the [eigenvector](https://www.geeksforgeeks.org/applications-of-eigenvalues-and-eigenvectors/) of matrix A for the corresponding eigenvalue.

It can also be written as :

where I am the identity matrix of the same shape as matrix A. And the above conditions will be true only if  will be non-invertible (i.e. singular matrix). That means,

From the above equation, we can find the eigenvalues \lambda, and therefore corresponding eigenvector can be found using the equation .

**How Principal Component Analysis(PCA) works?**

Hence, PCA employs a linear transformation that is based on preserving the most variance in the data using the least number of dimensions. It involves the following steps:

* Python3

|  |
| --- |
| **import** pandas as pd  **import** numpy as np    # Here we are using inbuilt dataset of scikit learn  **from** sklearn.datasets **import** load\_breast\_cancer    # instantiating  cancer **=** load\_breast\_cancer(as\_frame**=**True)  # creating dataframe  df **=** cancer.frame    # checking shape  **print**('Original Dataframe shape :',df.shape)    # Input features  X **=** df[cancer['feature\_names']]  print('Inputs Dataframe shape   :', X.shape) |

**Output**:

Original Dataframe shape : (569, 31)  
Inputs Dataframe shape : (569, 30)

Now we will apply the first most step which is to standardize the data and for that, we will have to first calculate the mean and standard deviation of each feature in the feature space.

* Python3

|  |
| --- |
| # Mean  X\_mean **=** X.mean()    # Standard deviation  X\_std **=** X.std()    # Standardization  Z **=** (X **-** X\_mean) **/** X\_std |

The [covariance](https://www.geeksforgeeks.org/mathematics-covariance-and-correlation/) matrix helps us visualize how strong the dependency of two features is with each other in the feature space.

* Python3

|  |
| --- |
| # covariance  c **=** Z.cov()    # Plot the covariance matrix  **import** matplotlib.pyplot as plt  **import** seaborn as sns  sns.heatmap(c)  plt.show() |

**Output**:

Now we will compute the [eigenvectors](https://www.geeksforgeeks.org/eigen-values/) and [eigenvalues](https://www.geeksforgeeks.org/eigen-values/) for our feature space which serve a great purpose in identifying the principal components for our feature space.

* Python3

|  |
| --- |
| eigenvalues, eigenvectors **=** np.linalg.eig(c)  print('Eigen values:\n', eigenvalues)  print('Eigen values Shape:', eigenvalues.shape)  print('Eigen Vector Shape:', eigenvectors.shape) |

**Output**:

Eigen values:  
 [1.32816077e+01 5.69135461e+00 2.81794898e+00 1.98064047e+00  
 1.64873055e+00 1.20735661e+00 6.75220114e-01 4.76617140e-01  
 4.16894812e-01 3.50693457e-01 2.93915696e-01 2.61161370e-01  
 2.41357496e-01 1.57009724e-01 9.41349650e-02 7.98628010e-02  
 5.93990378e-02 5.26187835e-02 4.94775918e-02 1.33044823e-04  
 7.48803097e-04 1.58933787e-03 6.90046388e-03 8.17763986e-03  
 1.54812714e-02 1.80550070e-02 2.43408378e-02 2.74394025e-02  
 3.11594025e-02 2.99728939e-02]  
Eigen values Shape: (30,)  
Eigen Vector Shape: (30, 30)

Sort the eigenvalues in descending order and sort the corresponding eigenvectors accordingly.

* Python3

|  |
| --- |
| # Index the eigenvalues in descending order  idx **=** eigenvalues.argsort()[::**-**1]    # Sort the eigenvalues in descending order  eigenvalues **=** eigenvalues[idx]    # sort the corresponding eigenvectors accordingly  eigenvectors **=** eigenvectors[:,idx] |

Explained variance is the term that gives us an idea of the amount of the total variance which has been retained by selecting the principal components instead of the original feature space.

* Python3

|  |
| --- |
| explained\_var **=** np.cumsum(eigenvalues) **/** np.sum(eigenvalues)  explained\_var |

**Output**:

array([0.44272026, 0.63243208, 0.72636371, 0.79238506, 0.84734274,  
 0.88758796, 0.9100953 , 0.92598254, 0.93987903, 0.95156881,  
 0.961366 , 0.97007138, 0.97811663, 0.98335029, 0.98648812,  
 0.98915022, 0.99113018, 0.99288414, 0.9945334 , 0.99557204,  
 0.99657114, 0.99748579, 0.99829715, 0.99889898, 0.99941502,  
 0.99968761, 0.99991763, 0.99997061, 0.99999557, 1. ])

**Determine the Number of Principal Components**

Here we can either consider the number of principal components of any value of our choice or by limiting the explained variance. Here I am considering explained variance more than equal to 50%. Let’s check how many principal components come into this.

* Python3

|  |
| --- |
| n\_components **=** np.argmax(explained\_var >**=** 0.50) **+** 1  n\_components |

**Output**:

2

**Project the Data onto the Selected Principal Components**

* Find the projection matrix, It is a matrix of eigenvectors corresponding to the largest eigenvalues of the covariance matrix of the data. it projects the high-dimensional dataset onto a lower-dimensional subspace
* The eigenvectors of the covariance matrix of the data are referred to as the principal axes of the data, and the projection of the data instances onto these principal axes are called the principal components.
* Python3

|  |
| --- |
| # PCA component or unit matrix  u **=** eigenvectors[:,:n\_components]  pca\_component **=** pd.DataFrame(u,                               index **=** cancer['feature\_names'],                               columns **=** ['PC1','PC2']                              )    # plotting heatmap  plt.figure(figsize **=**(5, 7))  sns.heatmap(pca\_component)  plt.title('PCA Component')  plt.show() |

**Output**:

* Then, we project our dataset using the formula:
* Dimensionality reduction is then obtained by only retaining those axes (dimensions) that account for most of the variance, and discarding all others.

*Finding Projection in PCA*

* Python3

|  |
| --- |
| # Matrix multiplication or dot Product  Z\_pca **=** Z @ pca\_component  # Rename the columns name  Z\_pca.rename({'PC1': 'PCA1', 'PC2': 'PCA2'}, axis**=**1, inplace**=**True)  # Print the  Pricipal Component values  print(Z\_pca) |

**Output**:

PCA1 PCA2  
0 9.184755 1.946870  
1 2.385703 -3.764859  
2 5.728855 -1.074229  
3 7.116691 10.266556  
4 3.931842 -1.946359  
.. ... ...  
564 6.433655 -3.573673  
565 3.790048 -3.580897  
566 1.255075 -1.900624  
567 10.365673 1.670540  
568 -5.470430 -0.670047  
[569 rows x 2 columns]

The eigenvectors of the covariance matrix of the data are referred to as the principal axes of the data, and the projection of the data instances onto these principal axes are called the principal components. Dimensionality reduction is then obtained by only retaining those axes (dimensions) that account for most of the variance, and discarding all others.

**PCA using Using Sklearn**

There are different libraries in which the whole process of the principal component analysis has been automated by implementing it in a package as a function and we just have to pass the number of principal components which we would like to have. Sklearn is one such library that can be used for the PCA as shown below.

* Python3

|  |
| --- |
| # Importing PCA  **from** sklearn.decomposition **import** PCA    # Let's say, components = 2  pca **=** PCA(n\_components**=**2)  pca.fit(Z)  x\_pca **=** pca.transform(Z)    # Create the dataframe  df\_pca1 **=** pd.DataFrame(x\_pca,                         columns**=**['PC{}'.                         format(i**+**1)  **for** i **in** range(n\_components)])  print(df\_pca1) |

**Output:**

PC1 PC2  
0 9.184755 1.946870  
1 2.385703 -3.764859  
2 5.728855 -1.074229  
3 7.116691 10.266556  
4 3.931842 -1.946359  
.. ... ...  
564 6.433655 -3.573673  
565 3.790048 -3.580897  
566 1.255075 -1.900624  
567 10.365673 1.670540  
568 -5.470430 -0.670047  
[569 rows x 2 columns]

We can match from the above Z\_pca result from it is exactly the same values.

* Python3

|  |
| --- |
| # giving a larger plot  plt.figure(figsize**=**(8, 6))    plt.scatter(x\_pca[:, 0], x\_pca[:, 1],              c**=**cancer['target'],              cmap**=**'plasma')    # labeling x and y axes  plt.xlabel('First Principal Component')  plt.ylabel('Second Principal Component')  plt.show() |

**Output:**

* Python3

|  |
| --- |
| # components  pca.components\_ |

**Output**:

array([[ 0.21890244, 0.10372458, 0.22753729, 0.22099499, 0.14258969,  
 0.23928535, 0.25840048, 0.26085376, 0.13816696, 0.06436335,  
 0.20597878, 0.01742803, 0.21132592, 0.20286964, 0.01453145,  
 0.17039345, 0.15358979, 0.1834174 , 0.04249842, 0.10256832,  
 0.22799663, 0.10446933, 0.23663968, 0.22487053, 0.12795256,  
 0.21009588, 0.22876753, 0.25088597, 0.12290456, 0.13178394],  
 [-0.23385713, -0.05970609, -0.21518136, -0.23107671, 0.18611302,  
 0.15189161, 0.06016536, -0.0347675 , 0.19034877, 0.36657547,  
 -0.10555215, 0.08997968, -0.08945723, -0.15229263, 0.20443045,  
 0.2327159 , 0.19720728, 0.13032156, 0.183848 , 0.28009203,  
 -0.21986638, -0.0454673 , -0.19987843, -0.21935186, 0.17230435,  
 0.14359317, 0.09796411, -0.00825724, 0.14188335, 0.27533947]])

**Advantages of Principal Component Analysis**

1. **Dimensionality Reduction**: Principal Component Analysis is a popular technique used for [dimensionality reduction](https://www.geeksforgeeks.org/dimensionality-reduction/), which is the process of reducing the number of variables in a dataset. By reducing the number of variables, PCA simplifies data analysis, improves performance, and makes it easier to visualize data.
2. **Feature Selection**: Principal Component Analysis can be used for [feature selection](https://www.geeksforgeeks.org/feature-selection-techniques-in-machine-learning/), which is the process of selecting the most important variables in a dataset. This is useful in machine learning, where the number of variables can be very large, and it is difficult to identify the most important variables.
3. **Data Visualization**: Principal Component Analysis can be used for [data visualization](https://www.geeksforgeeks.org/what-is-data-visualization-and-why-is-it-important/). By reducing the number of variables, PCA can plot high-dimensional data in two or three dimensions, making it easier to interpret.
4. **Multicollinearity**: Principal Component Analysis can be used to deal with [multicollinearity](https://www.geeksforgeeks.org/multicollinearity-in-data/), which is a common problem in a regression analysis where two or more independent variables are highly correlated. PCA can help identify the underlying structure in the data and create new, uncorrelated variables that can be used in the regression model.
5. **Noise Reduction**: Principal Component Analysis can be used to reduce the noise in data. By removing the principal components with low variance, which are assumed to represent noise, Principal Component Analysis can improve the signal-to-noise ratio and make it easier to identify the underlying structure in the data.
6. **Data Compression**: Principal Component Analysis can be used for data compression. By representing the data using a smaller number of principal components, which capture most of the variation in the data, PCA can reduce the storage requirements and speed up processing.
7. **Outlier Detection**: Principal Component Analysis can be used for outlier detection. [Outliers](https://www.geeksforgeeks.org/machine-learning-outlier/) are data points that are significantly different from the other data points in the dataset. Principal Component Analysis can identify these outliers by looking for data points that are far from the other points in the principal component space.

**Disadvantages of Principal Component Analysis**

1. **Interpretation of Principal Components**: The principal components created by Principal Component Analysis are linear combinations of the original variables, and it is often difficult to interpret them in terms of the original variables. This can make it difficult to explain the results of PCA to others.
2. **Data Scaling**: Principal Component Analysis is sensitive to the scale of the data. If the data is not properly scaled, then PCA may not work well. Therefore, it is important to scale the data before applying Principal Component Analysis.
3. **Information Loss**: Principal Component Analysis can result in information loss. While Principal Component Analysis reduces the number of variables, it can also lead to loss of information. The degree of information loss depends on the number of principal components selected. Therefore, it is important to carefully select the number of principal components to retain.
4. **Non-linear Relationships**: Principal Component Analysis assumes that the relationships between variables are linear. However, if there are non-linear relationships between variables, Principal Component Analysis may not work well.
5. **Computational Complexity**: Computing Principal Component Analysis can be computationally expensive for large datasets. This is especially true if the number of variables in the dataset is large.
6. **Overfitting**: Principal Component Analysis can sometimes result in [overfitting](https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/), which is when the model fits the training data too well and performs poorly on new data. This can happen if too many principal components are used or if the model is trained on a small dataset.

**Frequently Asked Questions (FAQs)**

**1. What is Principal Component Analysis (PCA)?**

*PCA is a dimensionality reduction technique used in statistics and machine learning to transform high-dimensional data into a lower-dimensional representation, preserving the most important information.*

**2. How does a PCA work?**

*Principal components are linear combinations of the original features that PCA finds and uses to capture the most variance in the data. In order of the amount of variance they explain, these orthogonal components are arranged.*

**3. When should PCA be applied?**

*Using PCA is advantageous when working with multicollinear or high-dimensional datasets. Feature extraction, noise reduction, and data preprocessing are prominent uses for it.*

**4. How are principal components interpreted?**

*New axes are represented in the feature space by each principal component. An indicator of a component’s significance in capturing data variability is its capacity to explain a larger variance.*

**5. What is the significance of principal components?**

*Principal components represent the directions in which the data varies the most. The first few components typically capture the majority of the data’s variance, allowing for a more concise representation.*