TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

SEMESTRE ENERO-JUNIO 2020

ING. SISTEMAS COMPUTACIONALES

DATOS MASIVOS

TEMA:

VARIANZA

EQUIPO:

16210958 - Aguirre Ibarra Jesus Armando

14212024 - Garcia Hernandez Victor David

PROFESOR:

JOSE CHRISTIAN ROMERO HERNANDEZ

TIJUANA B.C. A 28 DE FEBRERO DEL 2020

Variance

In probability theory and statistics, **variance** is the expectation of the squared deviation of a random variable from its mean. Informally, it measures how far a set of (random) numbers are spread out from their average value. Variance has a central role in statistics, where some ideas that use it include descriptive statistics, statistical inference, hypothesis testing, goodness of fit, and Monte Carlo sampling. Variance is an important tool in the sciences, where statistical analysis of data is common.

The variance of a random variable X is the expected value of the squared deviation from the mean of X, $\mu = \mathrm{E}[X]$:

$$\operatorname{Var}(X) = \mathrm{E}\left[(X - \mu)^2\right].$$

This definition encompasses random variables that are generated by processes that are discrete, continuous, neither, or mixed. The variance can also be thought of as the covariance of a random variable with itself:

$$\operatorname{Var}(X) = \operatorname{Cov}(X, X).$$

The variance is also equivalent to the second cumulant of a probability distribution that generates X. The variance is typically designated as $\operatorname{Var}(X)$, σ_X^2, or simply σ^2 (pronounced "sigma squared"). The expression for the variance can be expanded:

$$\begin{aligned}
\operatorname{Var}(X) &= \mathrm{E}\left[(X - \mathrm{E}[X])^2\right] \\
&= \mathrm{E}\left[X^2 - 2X\,\mathrm{E}[X] + \mathrm{E}[X]^2\right] \\
&= \mathrm{E}\left[X^2\right] - 2\,\mathrm{E}[X]\,\mathrm{E}[X] + \mathrm{E}[X]^2 \\
&= \mathrm{E}\left[X^2\right] - \mathrm{E}[X]^2
\end{aligned}$$

In other words, the variance of X is equal to the mean of the square of X minus the square of the mean of X. This equation should not be used for computations using floating point arithmetic because it suffers from catastrophic cancellation if the two components of the equation are similar in magnitude. There exist numerically stable alternatives.

Variance is the interconnection of Sub-Typing relationships which are either of complicated types or of their constituent types. *Variance* explains inheritance correlation of Types that have *parameters* or *arguments* within them. These types belongs to the *generic* classes, which takes a type like a parameter. In the presence of Variance one can create relations between complicated types and in its absence we won't be able to reiterate the abstraction class. The Scala Variances are of three types, which are as follows:

1. Covariant
2. Contravariant
3. Invariant

Some important points:

- In Scala collection's types can be constructed more securely through Variances.
- Variances can provide us with some extra adjustable advancements.
- It also helps in the development of authentic applications.
- Variances can be applied on any Scala Types like List, Sets, etc.

Types of Variances

Let's discuss each type in detail.

- **Covariant:** If a generic class has a type parameter T, then its Covariant notation will be [+T]. Suppose, we have two List types of Scala i.e, S and T. where, S is sub-type of T, then you can say that List[S] is also the sub-type of List[T]. If two types are related like this then they fall under the Covariant type. List[T] can be called as *Generic*.

Syntax:
List[+T]

Here, **T** is the type parameter and **+** is the symbol of Covariance.
Example:

filter_none
edit

play_arrow

brightness_4

```
// Scala program of covariant
// type

// Creating an abstract class
// for Student
abstract class Student
{
    def name: String
}

// Creating a sub-class Girls
// of Student
case class Girls(name: String) extends Student

// Creating a sub-class Boys
// of Student
case class Boys(name: String) extends Student

// Creating an Object Covariance
// that inherits main method of
// App
object Covariance extends App
{
    // Creating a method
    def Studentnames(students: List[Student]): Unit =
    {
        students.foreach { student =>
```

```
        // Displays students name
        println(student.name)
    }
    }

    // Assigning names
    val boys: List[Boys] = List(Boys("Kanchan"), Boys("Rahul"))
    val girls: List[Girls] = List(Girls("Nidhi"), Girls("Geeta"))

    // Accessing list of boys
    Studentnames(boys)

    // Accessing list of girls
    Studentnames(girls)
}
```

Output:

Kanchan

Rahul

Nidhi

Geeta

- Here, List of boys and girls both belongs to the List of students as they are its sub-type and so, here names of all the students are displayed when the Super-type Student is called.

  Note:

  - Abstract class is utilized here to apply covariance as it has List[+T] with it where, the type parameter T is covariant.
  - A trait *App* is used here to speedily change objects into workable programs.

- **Contravariant:** If a generic class has a type parameter T, then its Contravariant notation will be [-T]. Suppose, we have two List types of Scala i.e, S and T. where, S is sub-type of T, but List[T] is the sub-type of List[S]. If two types are related like this then they fall under the Contravariant type. It is opposite of covariant.

Syntax:
List[-T]

Here, **T** is the type parameter and **–** is the symbol of Contravariance.
Example:

filter_none
edit

play_arrow

brightness_4

```
// Scala program of Variance of
// Contravariant type

// abstract class with a contravariant
// type parameter
abstract class Show[-T]
{

    // Method for printing
    // type T
    def print(value: T): Unit
}

// A class structure
abstract class Vehicle
{
    def name: String
}

// Creating sub-class of Vehicle
case class Car(name: String) extends Vehicle

// Creating sub-class of class
// Show
class VehicleShow extends Show[Vehicle]
{
    def print(vehicle: Vehicle): Unit =
```

```
    // Displays name of the vehicle
    println("The name of the vehicle is: " + vehicle.name)
}

// Creating sub-class of class
// Show
class CarShow extends Show[Car]
{
    def print(car: Car): Unit =

    // Displays name of the car
    println("The name of the car is: " + car.name)
}

// Inheriting main method of
// the trait App
object Contravariance extends App
{

    // Assigning value to the name
    val newCar: Car = Car("Scorpio")

    // Defining a method that
    // prints the name
    def printnewCar(show: Show[Car]): Unit =
    {
        show.print(newCar)
    }
```

```
    // Creating objects

    val showcar: Show[Car] = new CarShow

    val showvehicle: Show[Vehicle] = new VehicleShow


    // Accessing name

    printnewCar(showcar)

    printnewCar(showvehicle)

}
```

Output:
The name of the car is: Scorpio

The name of the vehicle is: Scorpio

-

  It is Contravariant so, we are able to substitute Show[Vehicle] for Show[Car] and that's why both vehicle and car returns the same name.
- **Invariant:** In Scala, generic types are by default invariant. Suppose, we have two List types of Scala i.e, S and T. where, S is sub-type of T but List[T] and List[S] are not at all related, then they fall under the invariant type.
  Syntax:
  List[T]

  Here, we don't use any symbol for invariant relationship.
  Note: The classes like Array[T], ListBuffer[T], ArrayBuffer[T] etc. are mutable so, they have invariant type parameter, if we use invariant type parameters in inheritance relationship or sub-typing then we will get a compilation error.