Mouaz Ali
CSCI 3327

StatsLibrary Demo

The StatsLibrary and StatsLibraryTester classes are Java components designed to provide a comprehensive set of statistical and probabilistic functionalities. These classes facilitate common statistical operations, including measures of central tendency, distribution calculations, and set operations. The StatsLibrary class encapsulates various statistical methods, while the StatsLibraryTester class serves as a practical demonstration of the library's capabilities through test cases. In this document, I will be showing a demonstration of the various methods as well as providing console outputs of the test cases.

For context, there are two sets used for all of the central tendencies, standard deviation and variance, and set operations test cases, which are given below:

- Set A: {1, 3, 5, 6, 8, 8}
- Set B: {1, 2, 4, 7, 7}

Mean, Median, Mode

Methods:

```java
//Mean
public double mean(ArrayList<Integer> list) {
    double sum = 0;
    double avg = 0;
    for (int i = 0; i < list.size(); i++) {
        sum += list.get(i);
    }

    avg = sum / list.size();
    return avg;
}
```

```java
//Median
public double median(ArrayList<Integer> list) {
    Collections.sort(list);
    double median;

    if (list.size() % 2 == 1) {
        int odd = (list.size() - 1)/ 2;
        median = list.get(odd);
    }
    else {
        int even = list.size() / 2;
        median = (double) (list.get(even) + list.get(even - 1)) / 2;
    }

    return median;
}
```

```java
//Mode
public Integer mode(ArrayList<Integer> list) {
    // Returns null if there is no mode or if there is more than one mode
    HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
    int temp = 0;
    int greatest = 0;
    int index = 0;
    Integer mode;
    for (int i = 0; i < list.size(); i++) {
        int count = 0;
        for (int j = 0; j < list.size(); j++) {
            if (list.get(i) == list.get(j))
                count++;
        }
        map.put(list.get(i), count);
    }

    for (int i = 0; i < list.size(); i++) {
        temp = map.get(list.get(i));
        if (temp > greatest) {
            greatest = temp;
            index = i;
        }
    }
    map.remove(list.get(index));

    if (map.containsValue(greatest) || greatest <= 1)
        mode = null;
    else
        mode = list.get(index);

    return mode;
}
```

Output:

```
Mean (Set A): 5.166666666666667
Mean (Set B): 4.0
Median (Set A): 5.5
Median (Set B): 4.0
Mode (Set A): 8
Mode (Set B): null
```

Standard Deviation and Variance:

Methods:

```java
//Standard Deviation
public double standardDeviation(ArrayList<Integer> list) {
    double sum = 0;
    double standardDeviation = 0;
    double avg = mean(list);

    for (int i = 0; i < list.size(); i++) {
        sum += Math.pow((list.get(i) - avg), 2);
    }
    standardDeviation = Math.sqrt((sum/list.size()));
    return standardDeviation;
}
```

```java
// Variance
public double variance(ArrayList<Integer> list) {
    double avg = mean(list);
    double sumSquaredDifferences = 0;

    for (int i = 0; i < list.size(); i++) {
        sumSquaredDifferences += Math.pow((list.get(i) - avg), 2);
    }

    return sumSquaredDifferences / list.size();
}
```

Output:

```
Standard Deviation (Set A): 2.544056253745625
Variance (Set B): 5.2
```

Set Operations:

Methods:

```java
//Complement
public ArrayList<Integer> complement(ArrayList<Integer> list1, ArrayList<Integer> list2) {
    for (int i = 0; i < list1.size(); i++) {
        if (list2.contains(list1.get(i)))
            list2.remove(list1.get(i));
    }
    return list2;
}
```

```java
//Intersection
public ArrayList<Integer> intersection(ArrayList<Integer> list1, ArrayList<Integer> list2) {
    ArrayList<Integer> intersection = new ArrayList<Integer>();

    for (int i = 0; i < list1.size(); i++) {
        if (list2.contains(list1.get(i)))
            intersection.add(list1.get(i));
    }

    return intersection;
}
```

```java
//Union
public ArrayList<Integer> union(ArrayList<Integer> list1, ArrayList<Integer> list2) {
    for (int i = 0; i < list1.size(); i++) {
        if (!list2.contains(list1.get(i)))
            list2.add(list1.get(i));
    }

    return list2;
}
```

Output:

```
Complement (Set A, Universal Set): [2, 4, 7, 9, 10]
Intersection of Sets: [1, 6]
Union of Sets: [1, 2, 4, 6, 7, 3, 5, 8]
```

Factorial

BigInteger Method:

```java
//Factorial using BigInteger
public BigInteger factorial(int x) {
    BigInteger result = BigInteger.valueOf(x);

    for (int i = 1; i < x; i++)
        result = result.multiply(BigInteger.valueOf(i));

    return result;
}
```

Long method:

```java
// Factorial using long
public long factorialLong(int x) {
    if (x < 0) {
        throw new IllegalArgumentException("Input must be a non-negative integer.");
    }

    long result = 1;
    for (int i = 1; i <= x; i++) {
        result *= i;
    }

    return result;
}
```

Output:

```
Factorial (BigInteger) of 5: = 120
Factorial (Long) of 5: 120
```

Permutation and Combination

Permutation method:

```java
//Permutation
public BigInteger permutation(int n, int r) {
    BigInteger result;

    result = factorial(n).divide(factorial(n - r));

    return result;
}
```

Combination method:

```java
//Combination
public BigInteger combination(int n, int r) {
    BigInteger result;

    result = (factorial(n)).divide((factorial(r).multiply(factorial(n - r))));

    return result;
}
```

Output:

```
Permutation of 13 and 4 = 17160
Combination of 13 and 4 = 715
```

Dependency and Independency

Method:

```java
//Determining Dependency/Independency
public boolean Dependency(ArrayList<Integer> list1, ArrayList<Integer> list2) {
    // Check if there is any common element between the two lists
    for (Integer element : list1) {
        if (list2.contains(element)) {
            // There is a common element, so the lists are dependent
            return true;
        }
    }
    // No common element found, lists are independent
    return false;
}
```

Output:

```
Are Set A and B independent or dependent?: Dependent
```

Conditional Probability

Method:

```java
// Conditional Probability
public double conditionalProbability(ArrayList<Integer> eventA, ArrayList<Integer> eventB) {
    // Check if eventB has occurred, then calculate the probability of eventA given eventB
    if (eventB.isEmpty()) {
        System.out.println("Error: Event B is empty. Cannot calculate conditional probability.");
        return -1; // Return a negative value to indicate an error
    }

    ArrayList<Integer> intersectionAB = intersection(eventA, eventB);
    double probabilityB = (double) intersectionAB.size() / eventB.size();

    // Check if the probability of eventB is zero
    if (probabilityB == 0) {
        System.out.println("Error: Probability of Event B is zero. Cannot calculate conditional probability.");
        return -1; // Return a negative value to indicate an error
    }

    double probabilityAandB = (double) intersectionAB.size() / eventA.size();

    // Calculate conditional probability P(A|B) = P(A and B) / P(B)
    return probabilityAandB / probabilityB;
}
```

For this test case, assume that

Event A = (3, 6, 8)

Event B = (1, 2, 4, 6, 7)

Output:

```
Conditional Probability of A given B: 1.6666666666666665
```

Binomial Distribution

Methods:

```java
//Binomial Distribution
public double binomialDistribution(int n, int y, double p) {
    double probability;
    //q is the probability of failure for a single trial
    double q = 1 - p;
    double comb = combination(n, y).doubleValue();
    probability = comb * (Math.pow(p,  y)) * (Math.pow(q,  (n - y)));
    return probability;
}
```

```java
// Expected value for Binomial Distribution
public double binomialExpectedValue(int n, double p) {
    return n * p;
}
```

```java
// Variance for Binomial Distribution
public double binomialVariance(int n, double p) {
    return n * p * (1 - p);
}
```

Output:

```
Binomial Distribution (p=0.90, q=0.20, n=8, y=5): 0.0330674399
Binomial Distribution - Expected Value: 7.2
Binomial Distribution - Variance: 0.7199999999999999
```

Geometric Distribution

Methods:

```java
//Geometric Distribution
public double geometricDistribution(int n, double p) {
    double probability;
    //q is the probability of failure for a single trial
    double q = 1 - p;

    probability = (Math.pow(q, n-1)) * p;

    return probability;
}
```

```java
// Expected value for Geometric Distribution
public double geometricExpectedValue(double p) {
    return 1 / p;
}
```

```java
// Variance for Geometric Distribution
public double geometricVariance(double p) {
    return (1 - p) / (p * p);
}
```

Output:

```
Geometric Distribution (p=0.06, q=0.94, n=3): 0.053015999999
Geometric Distribution - Expected Value: 16.666666666666668
Geometric Distribution - Variance: 261.1111111111111
```

Hypergeometric Distribution

Methods:

```java
//Hypergeometric Distribution
public double hypergeometricDistribution(int n, int m, int r, int y) {
    double probability;

    probability = (combination(r, y).multiply(combination(n - r, m - y))).doubleValue() / (combination(n, m).doubleValue());

    return probability;
}
```

```java
// Expected value for Hypergeometric Distribution
public double hypergeometricExpectedValue(int n, int m, int r) {
    return (double) r * n / m;
}
```

```java
// Variance for Hypergeometric Distribution
public double hypergeometricVariance(int n, int m, int r) {
    return (double) r * (m - r) * n * (n - 1) / (m * m * (m - 1));
}
```

Output:

```
Hypergeometric Distribution (n=10, m=5, r=3, y=2): 0.4166
Hypergeometric Distribution - Expected Value: 6.0
Hypergeometric Distribution - Variance: 5.4
```

Negative Binomial Distribution

Methods:

```java
// Negative Binomial Distribution
public double negativeBinomialDistribution(int r, double p, int x) {
    if (r <= 0 || p <= 0 || p >= 1 || x < 0) {
        throw new IllegalArgumentException("Invalid parameters for negative binomial distribution.");
    }
    double q = 1 - p;
    double comb = combination(r + x - 1, x).doubleValue();
    return comb * Math.pow(p, r) * Math.pow(q, x);
}
```

```java
// Expected Value of Negative Binomial Distribution
public double negativeBinomialExpectedValue(int r, double p) {
    if (r <= 0 || p <= 0 || p >= 1) {
        throw new IllegalArgumentException("Invalid parameters for negative binomial distribution.");
    }
    return r / p;
}
```

```java
// Variance of Negative Binomial Distribution
public double negativeBinomialVariance(int r, double p) {
    if (r <= 0 || p <= 0 || p >= 1) {
        throw new IllegalArgumentException("Invalid parameters for negative binomial distribution.");
    }
    return r * (1 - p) / (p * p);
}
```

Output:

```
Negative Binomial Distribution (r=3, p=0.2, x=5): 0.05505
Negative Binomial Distribution - Expected Value: 15.0
Negative Binomial Distribution - Variance: 60.0
```