Applications of Data Analysis

Joel Lampikari 503542
Kasperi Keski-Loppi 503572

Exercise 6.1

For this exercise, we created a method that generates datasets depending on the input value sampleSize. The dataset consists of randomly generated integer values ranging from 0 to 100, which are then normalized. Then, each row of the dataset is given a "label" value of either 0 or 1. Lastly, the dataset is shuffled.

```
 9   def generateDataset(sampleSize):
10       dataset = []
11       normalized_column = []
12       for row in range(sampleSize):
13           col = []
14           col.append(random.randint(0, 100))
15           normalized_column.append(col[0])
16           if (row < sampleSize/2):
17               col.append(0)
18           else:
19               col.append(1)
20           dataset.append(col)
21
22       normalized_column = stats.zscore(normalized_column)
23       for row in range(sampleSize):
24           dataset[row][0] = normalized_column[row]
25       random.shuffle(dataset)
26
27       return dataset
```

Figure 1. generateDataset() method.

In the main method of the program, datasets are generated using different sample sizes: 10, 50, 100, 500 and 2000. For each sample size, new datasets are generated a 100 times to get 100 different c-index results.

```
samples = [10, 50, 100, 500, 2000]

print ''

for k in range(len(samples)):
    sample = samples[k]
    c_indexes_over_0_7 = 0
    c_indexes = []
    for x in range(0, 100):
        dataset = generateDataset(sample)
        predictions = []
        actual_labels = []
        for row in range(0, len(dataset)):
            testInstance = dataset[row]
```

Figure 2. The loop structure in the main method.

Results

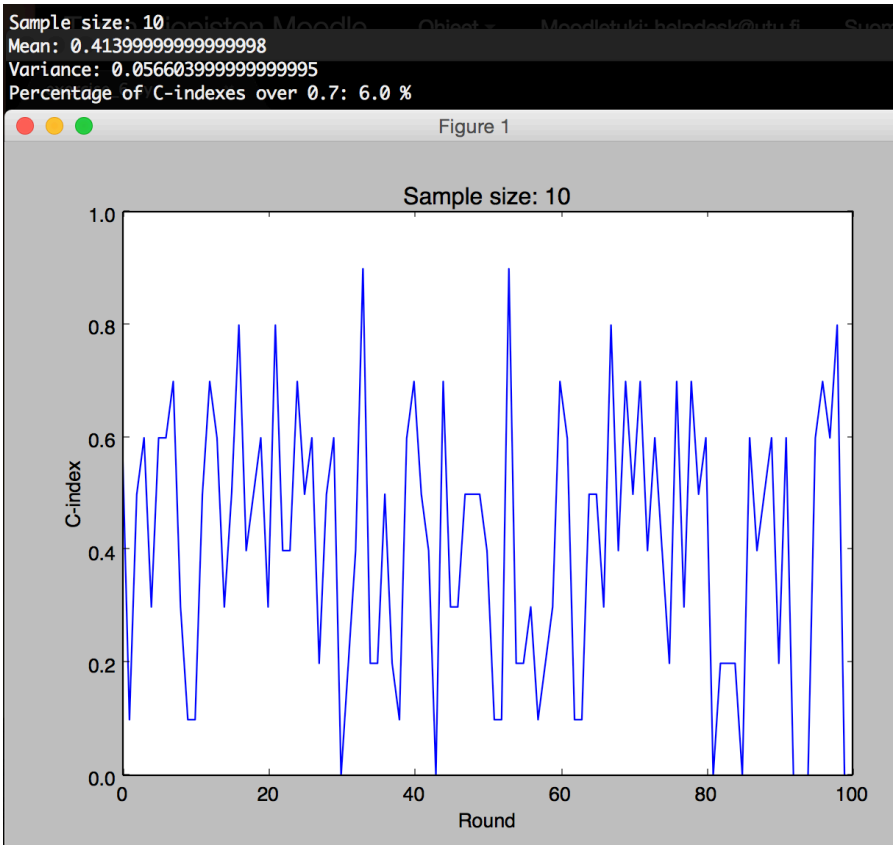

Figure 3. Results for sample size 10.

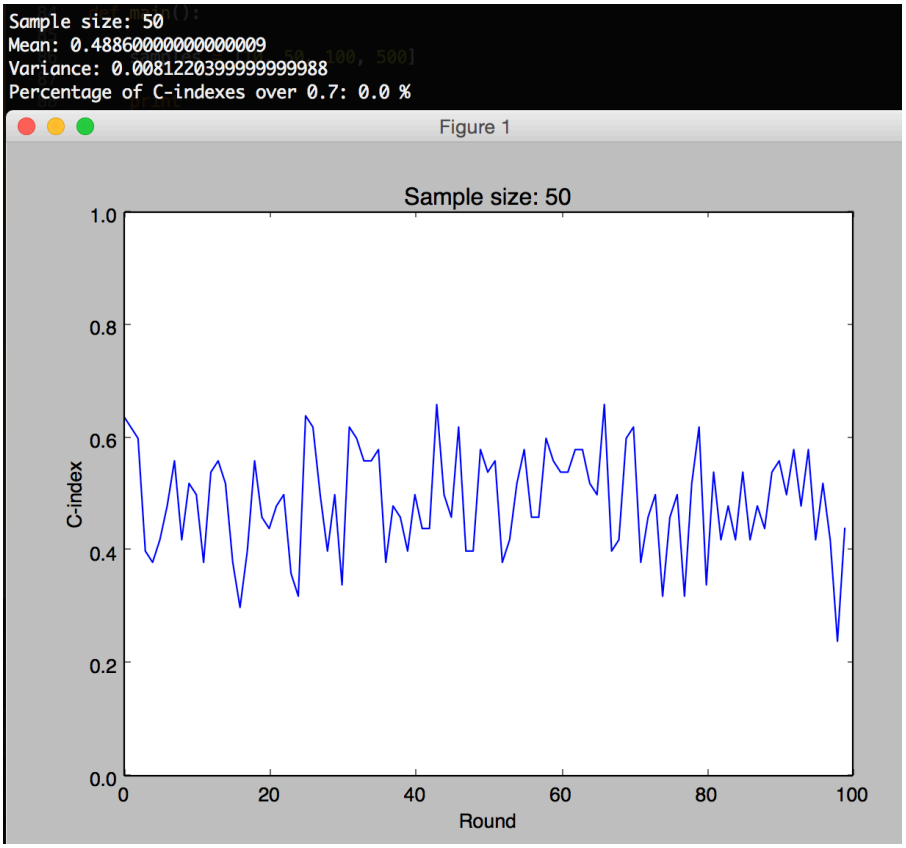

Figure 4. Results for sample size 50.

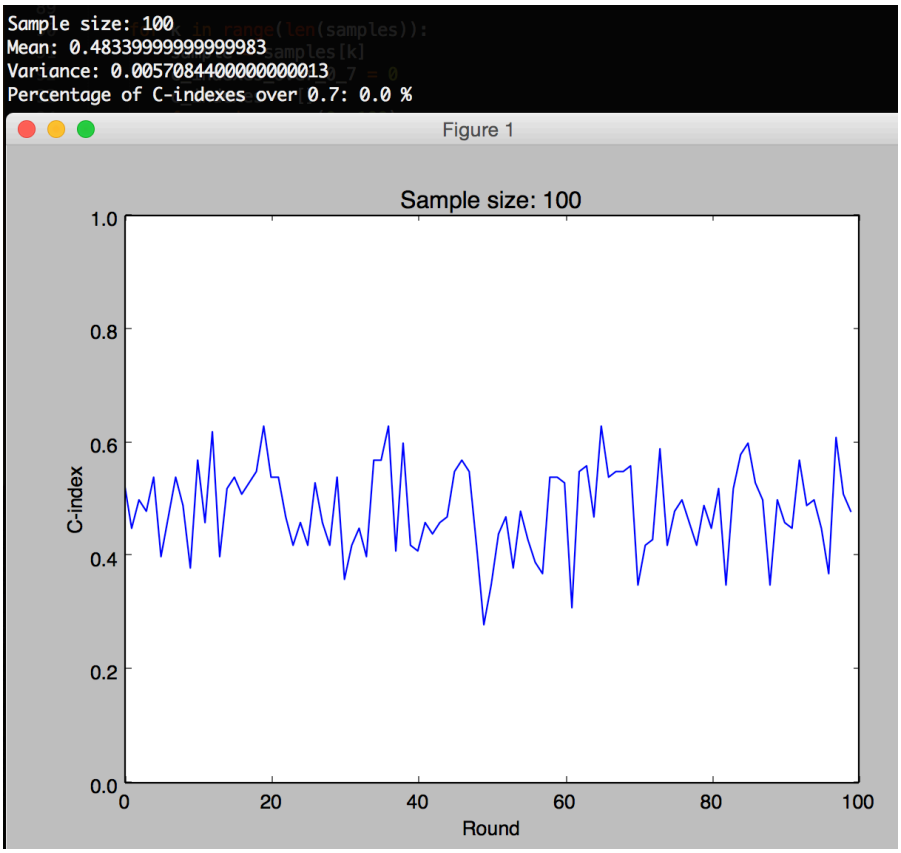

Figure 5. Results for sample size 100.

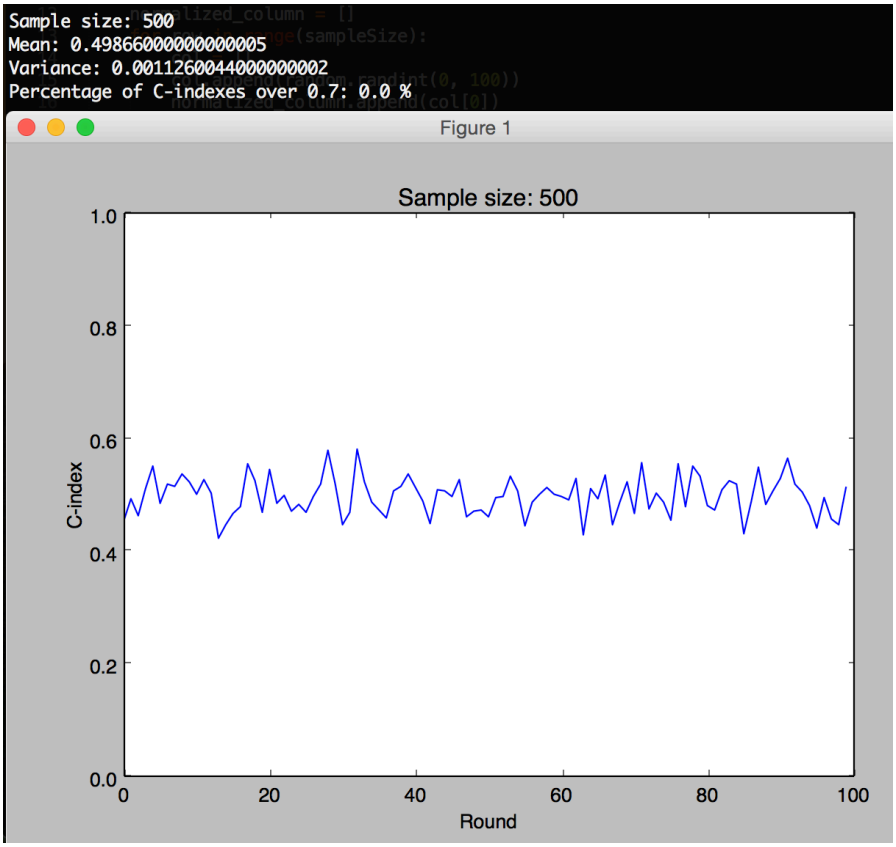

Figure 6. Results for sample size 500.

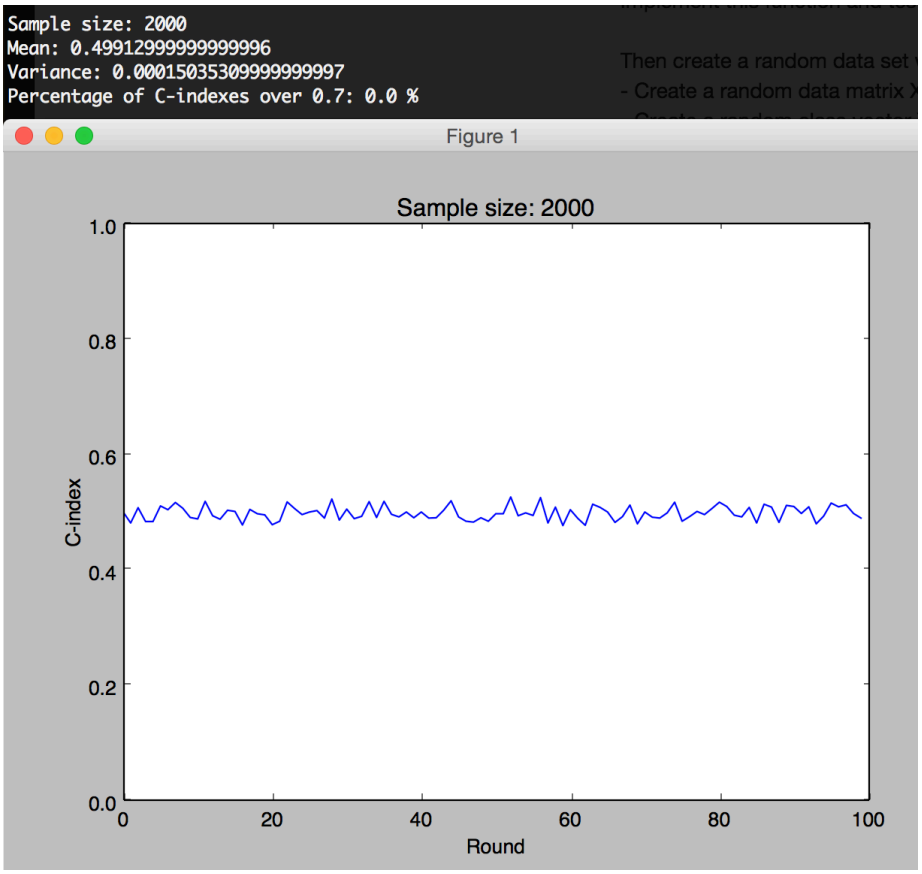

Figure 7. Results for sample size 2000.

As can be seen from the results, the variance of the C-indexes decreases drastically as the sample size grows, settling around the value of 0.5.

Exercise 6.2

For the second exercise, the dataset was generated with 50 samples, each consisting of 1000 features. Attributes were randomly generated integer values ranging from 0 to 100, which were then normalized. Then, each row of the dataset was given a "label" value of either 0 or 1. Lastly, the dataset was shuffled.

```
 9  def generateDataset(sampleSize):
10      dataset = np.zeros((sampleSize, 1001))
11      for col in range(1000):
12          column = []
13          for row in range(sampleSize):
14              column.append(random.randint(0, 100))
15          normalized_column = stats.zscore(column)
16          for row in range(sampleSize):
17              dataset[row][col] = normalized_column[row]
18      for row in range(len(dataset)):
19          if (row < sampleSize/2):
20              dataset[row][-1] = 0
21          else:
22              dataset[row][-1] = 1
23      np.random.shuffle(dataset)
24      return dataset
```

Figure 8. Method for generating a dataset.

For selecting the features, we created a method called selectFeatures(). It has the dataset (dataset) and the number of features to select (select_count) as its input values. It has two slightly different implementations: one that leaves out the test instance and one that does not. In both cases, it returns a reduced dataset that has features equal to the select_count input parameter.

```
80  def selectFeatures(dataset, select_count):
81      correlation_values = []
82      for col in range(len(dataset[0])-1):
83          attribute_col = []
84          label_col = []
85          for row in range(len(dataset)):
86              attribute_col.append(dataset[row][col])
87              label_col.append(dataset[row][-1])
88          tau, p_value = stats.kendalltau(attribute_col, label_col)
89          correlation_values.append([abs(tau), col])
90
91      reduced_dataset = np.zeros((50, select_count + 1))
92      correlation_values.sort(key=operator.itemgetter(0), reverse=True)
93      for row in range(len(dataset)):
94          for col in range(0, select_count):
95              reduced_dataset[row][col] = dataset[row][correlation_values[col][1]]
96      for row in range(len(reduced_dataset)):
97          reduced_dataset[row][-1] = dataset[row][-1]
98      return reduced_dataset
```

Figure 9. Method for feature selection, the version that includes the test instance.

6.2.1: The Wrong Way

For the wrong way, we left the feature selection outside the leave-one-out cross validation loop, i.e. the pre-processing is done before the cross validation.

```
reduced_dataset = selectFeatures(dataset, 10)

for row in range(0, len(reduced_dataset)):
    testInstance = reduced_dataset[row]
```

Figure 10. Feature selection done outside the loop.

Results

```
Round: 0
C-index: 0.37037037037037035

Round: 1
C-index: 0.6189300411522634

Round: 2
C-index: 0.349009900990099

Round: 3
C-index: 0.6450287592440427

Round: 4
C-index: 0.6291254125412541

Round: 5
C-index: 0.3583815028901734
```

Figure 11. C-index values for the wrong way. The program was run 6 times to gain an impression on how the algorithm behaves.

6.2.2: The Right Way

For the right way, we included the feature selection inside the leave-one-out cross validation loop – this way the feature selection is done separately for each round and the test instance is excluded from it.

```
for row in range(0, len(dataset)):

    reduced_dataset = selectFeatures(dataset, 10, row)
    testInstance = reduced_dataset[row]
```

Figure 12. Feature selection done on every round. Here the selectFeatures() method gets the corresponding row as an input parameter. It is used to exclude the test instance from the feature selection.

Results

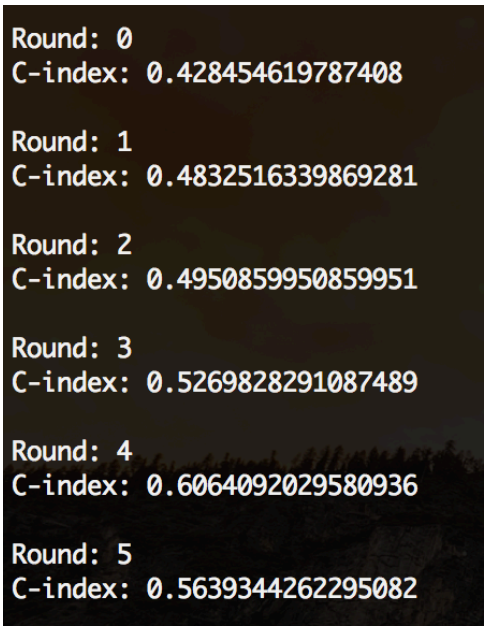

Figure 13. C-index values for the right way. The program was run 6 times to gain an imppression on how the algorithm behaves. It can be seen, that the values are closer to 0.5 than in the previous implementation.