1 Relative Häufigkeit mit BOA

Das gegebene Boa-Skript analysiert Codeprojekte, um die Häufigkeit der Kontrollstrukturen, `TRY` und `WITH`, zu messen und als Durchschnittswert pro Projekt auszugeben. Der Ablauf des Skripts lässt sich in drei Hauptteile unterteilen: die Eingabe- und Ausgabedefinition, die Erstellung eines Visitors zur Analyse von Statements sowie die Traversierung der Code-Repositories.

Code

```
p: Project = input;

list: output mean[string] of float;

counter := visitor {
    before node: Statement -> {
        if(node.kind == StatementKind.TRY or node.kind == StatementKind.WITH)
            list[p.name] << 1;
        else
            list[p.name] << 0;
    }
};

visit(p, visitor{
    before node: CodeRepository -> {
        snapshot := getsnapshot(node);
        foreach (i: int; def(snapshot[i]))
            visit(snapshot[i], counter);
    }
});
```

Erklärung

Zunächst wird das Projekt, das analysiert werden soll, über die Eingabevariable `p` definiert. Die Ausgabestruktur wird als eine Liste (`list`) festgelegt, in der der Mittelwert von Zahlenwerten gespeichert wird. Diese Liste verwendet als Schlüssel den Namen des Projekts (`p.name`) und als Wert eine Gleitkommazahl, die den Durchschnitt repräsentiert.
Ein Visitor mit dem Namen `counter` wird erstellt, um durch die einzelnen Statements im Code zu iterieren. Dieser Visitor überprüft in seinem `before`-Block, ob das aktuelle Statement ein `TRY`- oder `WITH`-Statement ist. Falls dies der Fall ist, wird der Wert `1` für das entsprechende Projekt in die Liste eingefügt. Andernfalls wird der Wert `0` hinzugefügt. Dieser Schritt ermöglicht es, die Verwendung der beiden Kontrollstrukturen quantitativ zu erfassen.

Im nächsten Teil des Skripts wird die Struktur des Projekts durchlaufen. Mithilfe der Funktion `getsnapshot` wird eine Momentaufnahme des Code-Repositories erstellt, die alle relevanten Bestandteile des Codes enthält. Anschließend wird iterativ jeder Teil des Snapshots durchlaufen, und der zuvor definierte Visitor `counter` wird auf diese Teile angewendet. Dadurch wird sichergestellt, dass alle relevanten Statements innerhalb des gesamten Repositories analysiert werden.
Am Ende sammelt die Liste `list` für jedes Projekt die entsprechenden Werte. Der Mittelwert, der durch das Schlüsselwort `mean` in der Definition der Liste berechnet wird, gibt an, wie häufig die `TRY`- und `WITH`-Anweisungen im Verhältnis zu anderen Statements innerhalb eines Projekts verwendet wurden. Dieses Ergebnis ist nützlich, um statistische Einblicke in die Nutzung bestimmter Kontrollstrukturen in verschiedenen Projekten zu erhalten.

1.1 Verhältnis 2000-2022

Das zweite Boa-Skript unterscheidet sich in mehreren Aspekten vom ersten. Die wichtigsten Unterschiede betreffen die Art und Weise, wie die Zeit berücksichtigt wird und wie die Ergebnisse gespeichert werden.

1. Speicherung der Ergebnisse über Jahre hinweg

Im ersten Skript wurde nur eine einfache Liste verwendet, um den Wert für jedes Projekt zu speichern. In diesem zweiten Skript wird jedoch eine zweidimensionale Liste verwendet, bei der zusätzlich zum Projektnamen auch das Jahr berücksichtigt wird. Dies ermöglicht es, die Ergebnisse für jedes Jahr separat zu speichern, und die Häufigkeit der `TRY`- und `WITH`-Statements wird für jedes Jahr des Projekts ermittelt.

```
list: output mean[string][int] of float;
```

Im Gegensatz zum ersten Skript, wo es nur eine einfache Dimension für den Projektnamen gab, ist hier eine zweite Dimension für das Jahr (`int`) hinzugefügt worden.

1.2 2. Berücksichtigung von Jahresdaten

Ein weiterer Unterschied ist, dass das zweite Skript Snapshots zu unterschiedlichen Zeitpunkten für jedes Jahr zwischen 2000 und 2022 erstellt, und zwar rückwärts von 2022 bis 2000. Diese Jahre werden verwendet, um die Nutzung von `TRY` und `WITH` im Zeitverlauf zu analysieren.

1.2.1 Codeänderung: Berechnung der Jahre und Snapshots

Die Schleife zur Berechnung und Auswahl der Jahre und der zugehörigen Snapshots unterscheidet sich stark vom ersten Skript. Statt eine Momentaufnahme einmalig für das gesamte Projekt zu nehmen, wird im zweiten Skript für jedes Jahr ein separater Snapshot erstellt und analysiert.

```
for (year: int = 22; year >= 0; year--){
    date = addyear(T"Dec 31, 2022, 10:00:00 AM", year);
    snapshot := getsnapshot(node, date);
    if (len(snapshot) > 0)
        foreach(i: int; def(snapshot[i]))
            visit(snapshot[i], counter);
}
```

Dieser Code blockiert für jedes Jahr (ab 2022 rückwärts) ein Datum und erstellt zu diesem Zeitpunkt eine Momentaufnahme des Repositories, um die Statements zu analysieren.

1.3 3. Nutzung der Jahresangabe in der Liste

Ein weiterer Unterschied ist, dass bei der Speicherung der Häufigkeit der Kontrollstrukturen nun zusätzlich das Jahr berücksichtigt wird. Die Liste speichert für jedes Jahr die Häufigkeit der Statements, die das Skript analysiert hat.

```
if(node.kind == StatementKind.TRY or node.kind == StatementKind.WITH)
    list[p.name][yearof(date)] << 1;
else
    list[p.name][yearof(date)] << 0;
```

Im Gegensatz zum ersten Skript, wo nur `list[p.name]` verwendet wurde, um die Häufigkeit der `TRY`- und `WITH`-Statements zu speichern, speichert das zweite Skript die Werte nun auch in Bezug auf das Jahr, das aus der Variablen `date` extrahiert wird.

2 Statistische Auswertung mit Python

2.1 Aufgabe 1: Statistischer Test für den Unterschied der Mittelwerte

Ausgehend vom Selection-Process-Tree aus der Vorlesung wählen wir für die Bestimmung, ob der Unterschied der Mittelwerte zwischen den Java- und Python-Projekten statistisch signifikant ist, den **Mann-Whitney-U-Test**. Als erstes wird beobachtet, dass wir von zwei Samples ausgehen. Zusätzlich eignet sich dieser Test, da wir davon ausgehen, dass die Verhältnisse (`Ratio`) in den beiden Datensätzen nicht normalverteilt sind. Der **Mann-Whitney-U-Test** prüft, ob zwei unabhängige Stichproben aus der gleichen Verteilung stammen. Dabei wird die Nullhypothese getestet; dass die Verteilungen der beiden Gruppen identisch sind.
Im Code wurde der Test wie folgt implementiert:

- Zunächst wurden die Daten für beide Projekte (Python und Java) eingelesen und die wichtigsten statistischen Kennzahlen wie Mittelwert und Varianz berechnet.
- Der Mann-Whitney-U-Test wurde mit der Funktion `stats.mannwhitneyu` aus dem Modul `scipy.stats` durchgeführt, wobei die Verhältnisse der beiden Gruppen als Eingabedaten verwendet wurden.
- Zusätzlich zur Durchführung des Tests wurde die Effektstärke in Form von *Cohen's d* berechnet. Diese misst die Größe des Unterschieds zwischen den Gruppen und wird wie folgt berechnet:

  $$\text{Cohen's d} = \frac{\mu_1 - \mu_2}{\sigma_{\text{pooled}}}$$

  Dabei ist μ_1 der Mittelwert der ersten Gruppe, μ_2 der Mittelwert der zweiten Gruppe und σ_{pooled} die gepoolte Standardabweichung.

Für eine Sample-Größe von 1000 ergab der Test folgende Werte:

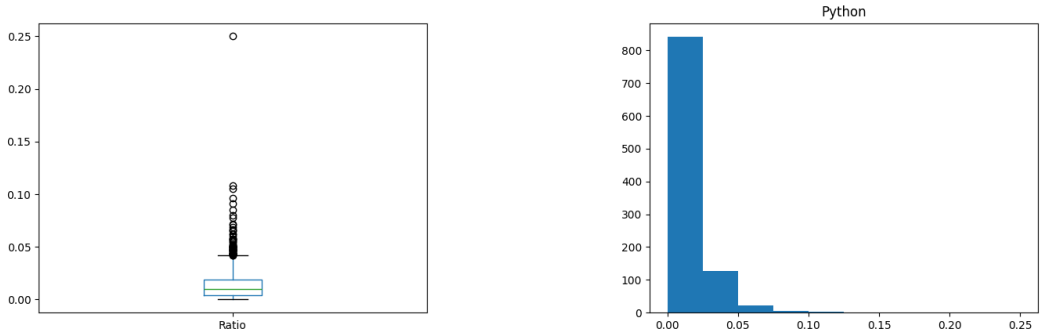
- **p-Wert**: 0.000003
- **Effektstärke (Cohen's d)**: 0.229

Da der p-Wert kleiner als das Signifikanzniveau von 1% ($\alpha = 0.01$) ist, wurde die Nullhypothese abgelehnt, was bedeutet, dass der Unterschied der Mittelwerte zwischen den Projekten Java und Python statistisch signifikant ist.

Nach der Konvention von Cohen werden Effektstärken wie folgt klassifiziert:

- $0.2 < d < 0.5$: kleiner Effekt,
- $0.5 < d < 0.8$: mittlerer Effekt,
- $d > 0.8$: großer Effekt.

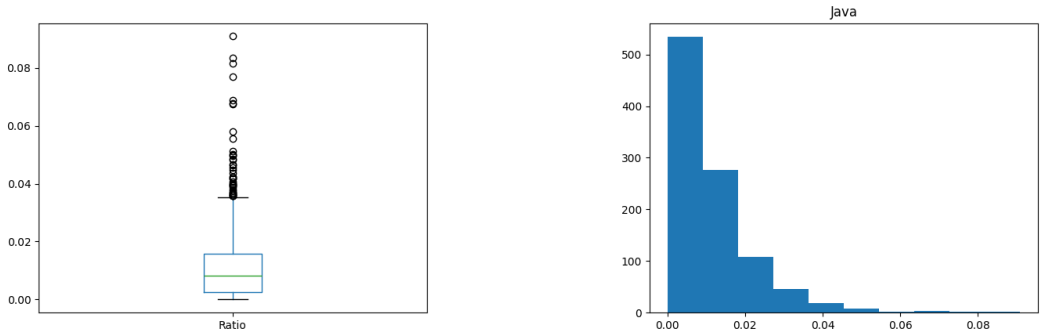
Der berechnete Wert gilt also als klein einzustufen.



(a) Boxplot der Verhältnisse für die Python-Daten

(b) Histogramm der Verhältnisse für die Python-Daten

Abbildung 1: Histogramm und Boxplot der Verhältnisse für Python (1000 Samples)



(a) Boxplot der Verhältnisse für die Java-Daten

(b) Histogramm der Verhältnisse für die Java-Daten

Abbildung 2: Histogramm und Boxplots der Verhältnisse für Java (1000 Samples)

2.2 Aufgabe 2: Bestimmung der Schranke für die Sample-Größe

Für die experimentelle Bestimmung der Schranke für die Stichprobengröße, ab der der Mann-Whitney-U-Test nicht mehr signifikant ist, wurde der Code so angepasst, dass verschiedene Stichprobengrößen getestet werden. Der Test wurde für Stichprobengrößen von 1 bis MAX durchgeführt, wobei für jede Größe der p-Wert berechnet wurde und der MAX-Wert die Gesamtgröße der Samples ist.
Durch einen einfachen Loop wurde die Schranke für die Sample-Größe wurde experimentell ermittelt und liegt bei einer Sample-Größe von 140 (Bei gleichbleibendem Seed für Samplegröße 1000). Ab dieser Größe wird der Test nicht mehr signifikant, was bedeutet, dass der p-Wert bei einer Größe von 140 p = 0.0104 beträgt, was größer als das Signifikanzniveau von 1% ist.

Aufgabe 3: Schlussfolgerungen für ähnliche Studien

In der Analyse wurde festgestellt, dass ab einer Stichprobengröße von $n = 140$ der Effekt nicht mehr statistisch signifikant ist.

1. Bedeutung der Stichprobengröße

Die Signifikanz eines Effekts hängt nicht nur von dessen Größe, sondern auch von der Stichprobengröße ab. Für kleine Effekte wie $d = 0.229$ ist eine ausreichend große Stichprobe erforderlich, um sie statistisch nachzuweisen. Studien mit zu kleinen Stichproben (z. B. $n < 140$) könnten fälschlicherweise zu dem Schluss kommen, dass kein Unterschied besteht.

2. Statistische Signifikanz vs. praktische Relevanz

Obwohl der Unterschied zwischen Python und Java in der Nutzung von `try`-Blöcken signifikant ist, hat er in der Praxis nur eine geringe Relevanz. Daher sollten zukünftige Studien nicht nur auf die Signifikanz (p-Wert), sondern auch auf die Effektstärke achten, um die praktische Bedeutung eines Ergebnisses besser bewerten zu können.

3. Robustheit der Analyse

Der Mann-Whitney-U-Test wurde verwendet, da er robust gegenüber nicht-normalverteilten Daten ist, was in diesem Fall zutrifft. Zusätzliche Maßnahmen, wie die Betrachtung von Konfidenzintervallen für die Effektstärke oder die Verteilung der Daten, könnten die Robustheit der Analyse weiter unterstützen.

Zusammenfassung

- Die Effektstärke $d = 0.229$ zeigt, dass die Unterschiede in der Nutzung von `try`-Blöcken klein sind.
- In der Praxis hat dieser Effekt jedoch nur begrenzte Bedeutung, da die Unterschiede relativ klein sind.
- Der Unterschied ist bei großen Stichproben statistisch signifikant ($p = 0.000003$), verliert jedoch bei kleineren Stichproben ($n < 140$) an Signifikanz.
- Für ähnliche Fragestellungen ist eine ausreichend große Stichprobe essenziell, um kleine Effekte nachzuweisen. Gleichzeitig sollte die praktische Relevanz kleiner Effekte kritisch hinterfragt werden.
- Große Stichproben, wie sie in dieser Analyse verwendet wurden ($n = 1000-10000$), ermöglichen es, auch kleine Effekte statistisch signifikant nachzuweisen. Dies sollte bei der Interpretation der Ergebnisse berücksichtigt werden.

Für vergleichbare Studien ergibt sich die Schlussfolgerung, dass die Signifikanz jedes Mal neu überprüft und berechnet werden sollte. Schon eine geringe Stichprobengröße kann statistisch signifikante Unterschiede offenbaren. Gleichzeitig kann eine große Stichprobengröße eine trügerische Sicherheit vermitteln, da sie nicht zwangsläufig zu statistisch signifikanten Ergebnissen führen muss, wie das angeführte Beispiel verdeutlicht.