Evaluation von Modellen zur Fehlervorhersage: Probleme und Lösungsmöglichkeiten

Thilo Mende, Rainer Koschke
Arbeitsgruppe Softwaretechnik
Fachbereich 3, Universität Bremen
http://www.informatik.uni-bremen.de/st/
{tmende,koschke}@informatik.uni-bremen.de

1 Einführung

Die Qualität von Software spielt eine immer wichtigere Rolle. Qualitätssichernde Maßnahmen, wie zum Beispiel Tests oder Code-Reviews, haben mittlerweile einen großen Anteil an den Kosten der Entwicklung. Techniken, die diese Aktivitäten kosteneffektiver gestalten, versprechen also ein hohes Einsparpotential.

Eine Möglichkeit zur Optimierung sind Modelle zur Fehlervorhersage, so genannte *Defect Prediction Models (DPM)*. Diese sollen fehleranfällige Bereiche eines Systems identifizieren, so dass diese besonders intensiv getestet werden können. In der Vergangenheit wurde eine Vielzahl solcher Modelle vorgestellt, oft mit viel versprechender Vorhersagegüte. Ein Überblick über existierende Arbeiten ist bei Arisholm et al. [1] zu finden.

Ein offenes Problem ist jedoch die Evaluation: Welche Maße sind geeignet, die Vorhersagequalität praxisnah zu bewerten? In diesem Beitrag fassen wir die Ergebnisse von zwei unserer vorhergehenden Studien zusammen, die dieses Thema näher untersuchen und Probleme in der Evaluation aufzeigen [3, 4].

2 Grundlagen

Defect Prediction Models bedienen sich Verfahren des statistischen Lernens, um mittels einer Menge von *unabhängigen Variablen* den Verlauf einer *abhängigen Variable* vorherzusagen. Als Lernverfahren kommen Regressionsmodelle sowie Verfahren des Data Mining, wie Decision Trees, in Frage. Die unabhängige Variablen können diverse Metriken sein, zum Beispiel klassische Code-Metriken oder Prozessmetriken. Die abhängige Variable ist in unserem Fall die Information über Fehler. Meist wird diese Informationen pro Datei aus Bug-Tracking und Versions-Kontrollsystemen extrahiert und bestimmt so die Granularität der Vorhersage.

Zur Evaluation der Modelle werden oft Maße aus der Klassifikation wie zum Beispiel *Recall* und *Precision* eingesetzt. Diese quantifizieren, welcher Anteil der fehlerhaften Dateien korrekt identifiziert wurde und wieviele der als fehlerhaft vorhergesagten Dateien tatsächlich fehlerhaft sind. Eine in der Praxis einfacher zu interpretierende Alternative ist die so genannte *Defect Detection Rate (ddr)*. Hierfür gehen wir davon aus, dass ein festes Budget für zusätzliche qualitätssichernde Maßnahmen zur Verfügung steht, zum Beispiel um 20 % des Codes zu testen oder zu begutachten. Nun benutzen wir ein DPM um 20 % des Codes auszuwählen und evaluieren, wie viele Fehler in diesem Code idealerweise gefunden werden könnten.

Ein Problem bei der Evaluation ist die Granularität: Precision, Recall und ddr gehen zunächst davon aus, dass die Kosten für zusätzliche Tests oder Reviews für alle Dateien gleich sind. Im folgenden Abschnitt zeigen wir, welchen Einfluss diese Annahme auf triviale Modelle hat.

Wir verwenden im Folgenden öffentlich verfügbare Datensätze aus dem PROMISE-Repository[1]. Diese umfassen 12 NASA-Projekte sowie 3 Versionen der Eclipse IDE. Mehr Informationen über die verwendeten Datensätze sind bei Mende et al. [4] zu finden.

3 Vorhersagegüte eines trivialen DPM

Ein triviales Vorhersagemodell besteht darin, Dateien nach absteigender Größe zu sortieren. Dann entspricht ddr mit einem Budget von 20 % der Anzahl der Fehler, die in den größten 20 % der Dateien enthalten sind.

Die Ergebnisse eines solchen Modells, gemessen in ddr, sind in Tabelle 1 zu finden. Zunächst fällt auf, dass unser triviales Modell einen erstaunlich hohen Anteil der Fehler identifizieren kann. Die Werte sind nicht wesentlich schlechter als viele der in der Literatur beschriebenen DPMs und klingen zunächst viel versprechend: im Mittel werden 60 % der Fehler in 20 % der Dateien identifiziert. Schaut man jedoch auf den Anteil der *Lines of Code* (LoC), der in diesen Dateien enthalten ist, wird klar, dass ein solches Modell wertlos ist. Die 20 % größten Dateien enthalten im Mittel 64 % der LoC; es ist also nicht verwunderlich, dass in etwa gleich viele Fehler gefunden werden.

Für die Evaluation von DPMs ergeben sich damit zwei mögliche Lösungsmöglichkeiten: Entweder man geht davon aus, dass die Kosten für die zusätzlichen qualitätssichernden Maßnahmen nicht

[1] http://promisedata.org

Datensatz	ddr	LoC	eddr
kc1	0.55	0.65	0.14
kc3	0.63	0.65	0.23
kc4	0.33	0.69	0.10
jm1	0.54	0.61	0.15
pc1	0.59	0.58	0.26
pc2	0.81	0.66	0.23
pc3	0.53	0.59	0.07
pc4	0.61	0.56	0.22
pc5	0.95	0.85	0.06
cm1	0.51	0.60	0.09
mc2	0.52	0.64	0.11
mw1	0.62	0.50	0.38
Eclipse 2.0	0.62	0.67	0.19
Eclipse 2.1	0.54	0.68	0.15
Eclipse 3.0	0.61	0.68	0.17
Mittelwert	0.60	0.64	0.17

Abbildung 1: Evaluation des trivialen DPMs: für 20 % der Dateien gemessen in ddr, der darin enthaltene Anteil des Quelltextes in Lines of Code (LoC), sowie für 20 % der LoC, gemessen in eddr.

von LoC abhängen, wie beispielsweise beim System- oder Integrationstest. Dann sollte man bei der Evaluation eines DPMs ein triviales Modell zum Vergleich heranziehen, um die praktischen Vorteile des komplizierteren Modells zu überprüfen. Ein detaillierter Vergleich mit einer Reihe von DPMs ist bei Mende et al. [3] zu finden; erstaunlicherweise ist das triviale Modell oft nicht signifikant schlechter.

Wenn die Kosten (auch nur annähernd) quantifiziert werden können, sollte man sie für die Evaluation und gegebenenfalls auch für die Vorhersage heranziehen. Diesen Aspekt betrachten wir im nächsten Abschnitt genauer.

4 Einfluss der Kosten auf die Evaluation

Für einige Maßnahmen lassen sich die Kosten pro Datei zumindest approximieren: Für Code-Reviews sind die Kosten vermutlich ungefähr proportional zu LoC [1], und auch für Unit-Tests spielt die Größe eine entscheidende Rolle [2]. In diesem Fall ist ein DPM nur dann kosteneffektiv einzusetzen, wenn es mehr Fehler identifizieren kann als eine zufällige Auswahl an LoC. Gemessen werden kann dies mit einer modifizierten Variante der ddr, der *effective defect detection rate (eddr)*. Diese bestimmt, welcher Anteil der Fehler in einem vorher bestimmten Anteil der LoC, im folgenden 20 %, gefunden werden könnte.

Die Ergebnisse für unser triviales Modell sind in der letzten Spalte von Tabelle 1 zu finden: Im Mittel werden in 20 % der LoC nur 17 % der Fehler gefunden. Eine zufällige Auswahl von Dateien die 20 % der LoC enthalten würde im Mittel schon 20 % der Fehler identifizieren.

Untersucht man gängige Modelle zur Fehlervorhersage mit einem Maß wie eddr, stellt man fest, dass viele von diesen Modellen kaum kosteneffektiv einzusetzen sind — unter der Annahme dass die Kosten für zusätzliche Maßnahmen zumindest teilweise von der Länge einer Datei abhängen. Eine mögliche Verbesserung lässt sich erreichen, indem die abhängige Variable geändert wird: Anstatt das Vorhandensein von Fehlern kann die Fehlerdichte vorhergesagt werden. In einer Untersuchung auf den hier verwendeten Datensätzen kamen wir zu dem Ergebnis, dass dies zu einer deutlichen Verbesserung der Vorhersagegüte führt. Jedoch sind auch diese Modelle weit von einer optimalen Vorhersage entfernt, so dass noch unklar ist, ob sie kosteneffektiv in der Praxis eingesetzt werden können [4].

5 Fazit

Die Evaluation von DPMs ist nicht einfach — eine Vielzahl von Faktoren spielt bei der Auswahl geeigneter Vorgehensweisen und Maße eine Rolle. Der Vergleich mit einem trivialen Modell ist in jedem Fall als Benchmark hilfreich. Bei der Auswahl geeigneter Evaluationsmaße ist der Einsatzzweck zu beachten: für welche Art von Aktivitäten sollen die Ergebnisse eingesetzt werden, und wie sind die Kosten pro Datei verteilt? Sowohl das Lernverfahren als auch die Evaluationsmaße können dann, wie in Abschnitt 4 beschrieben, angepasst werden.

Die hier beschriebenen Ergebnisse beziehen sich auf den einmaligen Einsatz eines DPM. Wenn ein solches im Rahmen der Wartung kontinuierlich eingesetzt werden soll, müssen vermutlich sowohl die Kriterien für ein triviales Modell als auch die Evaluationsmaße geeignet angepasst werden. Wie genau bleibt noch zu untersuchen.

Literatur

[1] Erik Arisholm, Lionel C. Briand, and Eivind B. Johannessen. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Journal of Systems Software*, 83:2–17, 2010.

[2] Magiel Bruntink and Arie van Deursen. An empirical study into class testability. *Journal of Systems and Software*, 79(9):1219–1232, 2006.

[3] Thilo Mende and Rainer Koschke. Revisiting the evaluation of defect prediction models. In *PROMISE '09: Proceedings of the 5th International Conference on Predictor Models in Software Engineering*, pages 1–10, New York, NY, USA, 2009.

[4] Thilo Mende and Rainer Koschke. Effort-aware defect prediction models. In *European Conference on Software Maintenance and Reengineering*, pages 109–118, 2010.