DeepL
Abonnieren Sie DeepL Pro, um größere Dateien zu übersetzen.
Weitere Informationen finden Sie unter www.DeepL.com/pro

Reduzierung von Merkmalen zur Verbesserung der Fehlervorhersage

Shivkumar Shivaji, E. James Whitehead, Jr., Ram Akella
Universität von Kalifornien Santa Cruz
{shiv,ejw,ram}@soe.ucsc.edu

Sunghun Kim
Universität für Wissenschaft und Technologie
Hongkong hunkim@cse.ust.hk

Abstrakt - In letzter Zeit **haben sich Klassifizierer für maschinelles Lernen als eine Möglichkeit herauskristallisiert, das Vorhandensein eines Fehlers in einer an einer Quellcodedatei vorgenommenen Änderung vorherzusagen. Der Klassifikator wird zunächst auf Software-Historiendaten trainiert und dann zur Vorhersage von Fehlern verwendet. Zwei Nachteile bestehender klassifikatorbasierter Fehlervorhersagen sind die für den praktischen Einsatz möglicherweise unzureichende Genauigkeit und die Verwendung einer großen Anzahl von Merkmalen. Diese große Anzahl von Merkmalen wirkt sich negativ auf die Skalierbarkeit und Genauigkeit des Ansatzes aus. In diesem Beitrag wird eine Technik zur Auswahl von Merkmalen für die klassifikatorbasierte Fehlervorhersage vorgeschlagen. Diese Technik wird zur Vorhersage von Fehlern in Softwareänderungen angewandt, und die Leistung von Na¨ıve Bayes- und Support Vector Machine (SVM)-Klassifikatoren wird charakterisiert.**

Index Begriffe-Zuverlässigkeit**; Fehlervorhersage; Maschinelles Lernen; Merkmalsauswahl**

I. EINFÜHRUNG

Klassifikatoren können, wenn sie auf historischen Softwareprojektdaten trainiert werden, zur Vorhersage des Vorhandenseins eines Fehlers in einer einzelnen Softwareänderung auf Dateiebene verwendet werden, wie in früheren Arbeiten des zweiten und vierten Autors [1] (im Folgenden Kim et al. genannt), in Arbeiten von Hata et al. [2] und anderen gezeigt wurde. Der Klassifikator wird zunächst anhand von Informationen aus historischen Änderungen trainiert und kann dann dazu verwendet werden, eine neue Änderung entweder als fehlerhaft (Vorhersage, dass sie einen Fehler enthält) oder als sauber (Vorhersage, dass sie keinen Fehler enthält) zu klassifizieren. Obwohl diese Ergebnisse zu den besten Algorithmen zur Vorhersage von Fehlern gehören, sind sie vielleicht nicht stark genug, um in der Praxis verwendet zu werden.

Wir stellen uns eine Zukunft vor, in der Software-Ingenieure in ihrer Entwicklungsumgebung über Funktionen zur Vorhersage von Fehlern verfügen. Software-Ingenieure werden von einem Klassifizierer eine Rückmeldung darüber erhalten, ob jede Änderung, die sie vornehmen, fehlerhaft oder sauber ist. In jüngster Zeit haben wir einen Prototyp entwickelt, der vom Server berechnete Fehlervorhersagen innerhalb der Eclipse IDE anzeigt [3]. Ein Bug Prediction Service muss hochpräzise Vorhersagen liefern. Wenn Ingenieure einem Bug Prediction Service vertrauen sollen, muss er sehr wenige "Fehlalarme" liefern, also Änderungen, die als fehlerhaft vorhergesagt werden, aber in Wirklichkeit sauber sind. Wenn zu viele saubere Änderungen fälschlicherweise als fehlerhaft vorhergesagt werden, werden die Entwickler das Vertrauen in das Fehlervorhersagesystem verlieren.

Der von Kim et al. verwendete Ansatz zur Vorhersage von Fehlern im Rahmen der Änderungsklassifizierung beinhaltet die Extraktion von "Merkmalen" (im Sinne des maschinellen Lernens, die sich von Softwaremerkmalen unterscheiden) aus der Historie der an einem Softwareprojekt vorgenommenen Änderungen. Diese Merkmale umfassen alles, was durch Leerzeichen getrennt ist, im Code, der bei einer Änderung hinzugefügt oder gelöscht wurde. Dies führt zu einer großen Anzahl von Merkmalen, die in die Tausende oder sogar Zehntausende geht. Bei größeren Projektverläufen, die sich über Tausende von Revisionen oder mehr erstrecken, kann sich dies auf Hunderttausende von Features ausdehnen.

Die große Anzahl von Merkmalen hat ihren Preis. Die Hinzufügung vieler nicht nützlicher Merkmale verringert die Genauigkeit eines Klassifikators. In der Regel steigt die für die Klassifizierung benötigte Zeit mit der Anzahl der Merkmale und erreicht bei Zehntausenden von Merkmalen mehrere Sekunden pro Klassifizierung und bei großen Projektverläufen Minuten.

Ein Standardansatz (in der Literatur zum maschinellen Lernen) für den Umgang mit großen Merkmalsmengen besteht darin, einen Merkmalsauswahlprozess durchzuführen, um die Teilmenge von Merkmalen zu ermitteln, die die besten Klassifizierungsergebnisse liefert. In diesem Beitrag wird ein Merkmalsauswahlprozess vorgestellt, bei dem Merkmale mit dem niedrigsten Verstärkungsverhältnis verworfen werden, bis eine optimale Klassifizierungsleistung für ein bestimmtes Leistungsmaß erreicht ist.

In diesem Papier werden die folgenden Forschungsfragen untersucht.

Frage 1. Welche Möglichkeiten führen zu einer optimalen Fehlerprognose durch Merkmalsauswahl?
Die beiden Variablen, die sich auf die Leistung der Fehlervorhersage auswirken, werden in diesem Papier untersucht: (1) die Art des Klassifikators (Na¨ıve Bayes, Support Vector Machine) und (2) die für die Klassifizierung optimierte Metrik (recall, F-measure). Die Ergebnisse werden in Abschnitt IV-A dargestellt.

Die Ergebnisse für Frage 1 werden als Durchschnittswerte für alle Projekte im Korpus angegeben. In der Praxis ist es jedoch nützlich, die Bandbreite der Ergebnisse in einer Reihe von Projekten zu kennen. Dies führt zu unserer zweiten Frage.

Frage 2. Bereich der Fehlerleistung bei Verwendung der Merkmalsauswahl. Wie groß ist der Leistungsbereich des besten Bayes'schen (F-Maß-optimierten) Klassifikators über alle Projekte hinweg bei Verwendung der Merkmalsauswahl? (siehe Abschnitt IV-B)

Der primäre Beitrag dieses Papiers ist der Prozess der Verwendung von Gain Ratio für die Merkmalsauswahl, zusammen mit der Charakterisierung der Ergebnisse der Fehlervorhersage, die bei der Verwendung der Merkmalsauswahl erzielt werden. Ein Vergleich der Ergebnisse dieser Arbeit mit denen verwandter Arbeiten (siehe Abschnitt V) zeigt, dass die Änderungsklassifikation mit Merkmalsselektion andere bestehende klassifikationsbasierte Ansätze zur Fehlerprognose übertrifft. Darüber hinaus liegt die durchschnittliche Fehlerpräzision bei der Verwendung von Na¨ıve Bayes (F-Maß optimiert) bei 0,96, was darauf hindeutet, dass die Fehlervorhersagen im Allgemeinen hochpräzise sind, wodurch das Problem der "falschen Negative" vermieden wird. Im weiteren Verlauf des Papiers geben wir zunächst einen Überblick über den Ansatz der Änderungsklassifikation zur Vorhersage von Fehlern und erläutern dann den neuen Algorithmus für die Merkmalsauswahl (Abschnitt II). Anschließend beschreiben wir den experimentellen Kontext, einschließlich unseres Datensatzes, und die spezifischen Klassifikatoren (Abschnitt III). Die Bühne ist nun bereitet, und in den folgenden Abschnitten gehen wir auf die oben beschriebenen Forschungsfragen ein (Abschnitte IV-A

- IV-B). Das Papier endet mit einer Zusammenfassung verwandter Arbeiten (Abschnitt V) und der Schlussfolgerung.

II. KLASSIFIZIERUNG ÄNDERN

Die wichtigsten Schritte bei der Klassifizierung von Änderungen an einem einzelnen Projekt werden im Folgenden beschrieben:

Erstellen eines Korpus:

1. Änderungen auf Dateiebene werden aus der Revisionshistorie eines Projekts extrahiert, die in seinem SCM-Repository gespeichert ist (siehe Abschnitt II-A).
2. Die Bugfix-Änderungen für jede Datei werden durch die Untersuchung von Schlüsselwörtern in SCM-Änderungsprotokollen identifiziert (Abschnitt II-A).
3. Die fehlereinführenden und bereinigenden Änderungen auf Dateiebene werden durch Rückverfolgung der Revisionshistorie von Fehlerbehebungsänderungen (Abschnitt II-A) ermittelt.
4. Merkmale werden aus allen Änderungen extrahiert, sowohl aus fehlerhaften als auch aus sauberen. Zu den Merkmalen gehören alle Begriffe im gesamten Quellcode, die bei jeder Änderung geänderten Zeilen (Delta) und Änderungsmetadaten wie Autor und Änderungszeit. Falls verfügbar, werden in diesem Schritt auch Komplexitätsmetriken verwendet. Einzelheiten zu diesen Merkmalsextraktionstechniken werden in Abschnitt II-B beschrieben.

Alle Schritte bis zu diesem Punkt sind die gleichen wie in Kim et al. Der folgende Schritt ist der neue Beitrag in diesem Papier.

Auswahl der Merkmale:

5. Durchführen eines Merkmalsauswahlprozesses, der Gain Ratio verwendet, um einen reduzierten Satz von Merkmalen zu berechnen, wie in Abschnitt II-C beschrieben. Bei jeder Iteration der Merkmalsauswahl wird die Klassifikatorleistung für eine Metrik (in der Regel F-Maß oder Genauigkeit) optimiert. Die Merkmalsauswahl wird iterativ durchgeführt, bis optimale Punkte erreicht sind. Am Ende von Schritt 5 gibt es eine reduzierte Merkmalsmenge, die für die gewählte Klassifikatormetrik optimal funktioniert.

Einstufung:

6. Anhand der reduzierten Merkmalsmenge wird ein Klassifizierungsmodell trainiert. Obwohl viele Klassifizierungsverfahren eingesetzt werden können, konzentriert sich dieses Papier auf die Verwendung von Na¨ıve Bayes und SVM.
7. Sobald ein Klassifikator trainiert wurde, ist er einsatzbereit. Neue Änderungen können nun in den Klassifikator eingespeist werden, der feststellt, ob eine neue Änderung eher einer fehlerhaften Änderung oder einer sauberen Änderung ähnelt.

A. Finden von fehlerhaften und sauberen Änderungen

Um fehlereinleitende Änderungen zu finden, müssen Fehlerbehebungen zunächst durch Auswertung von Änderungsprotokollen identifiziert werden. Wir verwenden zwei Ansätze: die Suche nach Schlüsselwörtern in Logmeldungen wie "Fixed", "Bug" [4] oder anderen Schlüsselwörtern, die wahrscheinlich in einer Fehlerbehebung vorkommen, und die Suche nach Verweisen auf Fehlerberichte wie "#42233". Auf diese Weise können wir feststellen, ob eine gesamte Codeänderungstransaktion eine Fehlerbehebung enthält. Wenn dies der Fall ist, müssen wir die spezifische Datei-Änderung identifizieren, die den Fehler eingeführt hat. Bei den in dieser Arbeit untersuchten Systemen haben wir manuell überprüft, ob es sich bei den identifizierten Fix-Commits tatsächlich um Bugfixes handelt. Für JCP wurden alle Fehlerbehebungen mit Hilfe einer Quellcode-Fehleranalyse identifiziert.

Tracking-System-Haken. Dadurch waren wir bei JCP nicht mehr auf Änderungsprotokollmeldungen angewiesen.

Der von Śliwerski, Zimmermann und Zeller vorgeschlagene Algorithmus zur Identifizierung fehlereinleitender Änderungen (SZZ-Algorithmus)
[5] wird in der vorliegenden Arbeit verwendet. Nach der Identifizierung von Fehlerbehebungen verwendet SZZ ein Differenzwerkzeug, um festzustellen, was sich in den Fehlerbehebungen geändert hat.

B. Merkmalsextraktion

Eine Datei-Änderung umfasst zwei Quellcode-Revisionen (eine alte und eine neue Revision) und ein Änderungsdelta, das den hinzugefügten Code (hinzugefügtes Delta) und den gelöschten Code (gelöschtes Delta) zwischen den beiden Revisionen aufzeichnet. Eine Datei-Änderung hat zugehörige Metadaten, darunter das Änderungsprotokoll, den Autor und das Übergabedatum. Jeder Begriff in den Texten des Quellcodes, des Änderungsdeltas und des Änderungsprotokolls wird als Merkmal verwendet.

Wir sammeln acht Merkmale aus den Änderungsmetadaten: Autor, Commit-Stunde, Commit-Tag, kumulative Änderungsanzahl, kumulative Fehleranzahl, Länge des Änderungsprotokolls, geänderte LOC (hinzugefügte Delta-LOC + gelöschte Delta-LOC) und LOC des Quellcodes der neuen Revision.

Wir berechnen eine Reihe von traditionellen Komplexitätsmetriken des Quellcodes mit den Understand C/C++ und Java Tools [6]. Zur Generierung von Merkmalen aus dem Quellcode verwenden wir eine modifizierte Version des Bag-of-Words-Ansatzes (BOW) [7], genannt BOW+, die zusätzlich zu allen von BOW erfassten Begriffen auch Operatoren extrahiert, da wir glauben, dass Operatoren wie !=, ++ und && wichtige Begriffe im Quellcode sind. Wir führen die BOW+-Extraktion für hinzugefügte Deltas, gelöschte Deltas und neue Revisionen durch. Quellcode.

C. Algorithmus zur Merkmalsauswahl

Die Anzahl der in der Merkmalsextraktionsphase gesammelten Merkmale ist recht groß und reicht von 6.127 für Plone bis 41.942 für JCP (Tabelle I). Solch große Merkmalsmengen führen zu längeren Trainings- und Vorhersagezeiten und erfordern große Mengen an Speicherplatz für die Klassifizierung. Eine gängige Lösung für dieses Problem ist der Prozess der Merkmalsauswahl, bei dem nur die Teilmenge der Merkmale verwendet wird, die für die Klassifizierungsentscheidungen am nützlichsten sind.

Das primäre Werkzeug, das in dieser Arbeit zur Bestimmung der nützlichsten Merkmale verwendet wird, ist die auf dem Gain Ratio basierende Merkmalsauswahl. Gain Ratio verbessert den Informationsgewinn [8], ein bekanntes entropiebasiertes Maß für den Informationsbeitrag eines bestimmten Merkmals zu einer Klassifizierungsentscheidung.

Die Gain Ratio spielt die gleiche Rolle wie der Informationsgewinn, liefert aber stattdessen ein normalisiertes Maß für den Beitrag eines Merkmals zu einer Klassifizierungsentscheidung [8]. Wir haben festgestellt, dass Gain Ratio eine der leistungsfähigsten Techniken zur Auswahl von Merkmalen bei der Fehlervorhersage ist, nachdem wir viele andere untersucht haben. Weitere Einzelheiten über die Berechnung des auf Entropie basierenden Maßes für Gain Ratio und seine Funktionsweise können in einem einführenden Data-Mining-Buch nachgelesen werden, z. B. in [8].

Gain Ratio wird in einem iterativen Prozess verwendet, bei dem schrittweise kleinere Merkmalsmengen ausgewählt werden, wie in Algorithmus

1. Der Algorithmus zur Merkmalsauswahl beginnt mit dem Ausschneiden der

Algorithmus 1 Algorithmus zur Auswahl von Merkmalen für ein Projekt

1) Beginnen Sie mit allen Merkmalen, *F*
2) Berechnen Sie das Verstärkungsverhältnis über *F* und wählen Sie die oberen 50 % der Merkmale mit dem besten Verstärkungsverhältnis, *F/2*
3) Ausgewählte Merkmale, *selF = F/2*
4) Während |selF | ≥ 0,1%|F |, Schritte (a)-(d) durchführen
   a) Berechnung und Speicherung der fehlerfreien und fehlerhaften Präzision, der Wiedererkennung, der Genauigkeit, des F-Maßes und der ROC-AUC unter Verwendung eines Klassifizierers für maschinelles Lernen (z. B. Na¨ıve Bayes oder SVM) mit 10-facher Kreuzvalidierung
   b) Berechnen des Verstärkungsverhältnisses über *selF*
   c) Identifizieren Sie *removeF* , die 10 % der Merkmale von *selF* mit dem niedrigsten Verstärkungsverhältnis. Dies sind die am wenigsten nützlichen Merkmale in dieser Iteration.
   d) *selF = selF - removeF*
5) Bestimmen Sie für eine bestimmte Klassifikator-Metrik (z. B. Genauigkeit, F-Maß) das beste Ergebnis aus Schritt 4.a. Der Prozentsatz der Merkmale, der das beste Ergebnis liefert, ist für die gegebene Metrik optimal.

Die anfängliche Merkmalsmenge wird halbiert, um den Speicher- und Verarbeitungsbedarf für den Rest des Algorithmus zu verringern. Da optimale Feature-Sets in der Regel bei weniger als 10 % aller Features gefunden werden, reduziert dies die Iterationen des Algorithmus. Projekte mit einer großen Anzahl von Merkmalen können bei der anfänglichen Merkmalsauswahl noch aggressiver vorgehen; für PostgreSQL wurden nur 25 % und für JCP nur 12,5 % aller Merkmale für die anfängliche *Auswahl* verwendet.

In der Iterationsphase werden bei jeder Iteration die 10 % der verbleibenden Merkmale ermittelt, die für die Klassifizierung am wenigsten nützlich sind, und eliminiert (wenn wir stattdessen jeweils ein Merkmal reduzieren würden, wäre dieser Schritt ähnlich wie die rückwärtige Merkmalsauswahl [9]). Der Vorteil der von uns vorgeschlagenen Anzahl von 10 % der Merkmale auf einmal ist die verbesserte Geschwindigkeit des Algorithmus bei gleichbleibender Ergebnisqualität. So beginnt *selF* zum Beispiel mit 50 % aller Merkmale, wird dann auf 45 % aller Merkmale reduziert, dann auf 40,5 % und so weiter. In jedem Schritt wird die Fehlervorhersage der Änderungsklassifikation mit *selF* über die gesamte Revisionshistorie durchgeführt, wobei eine 10-fache Kreuzvalidierung verwendet wird, um die Möglichkeit einer Überanpassung an die Daten zu verringern.

Diese Iteration ist beendet, wenn nur noch wenige Merkmale übrig sind. Zu diesem Zeitpunkt gibt es eine Liste von Tupeln (Merkmal %, Klassifikatorbewertungsmetrik). Der letzte Schritt besteht darin, diese Liste zu durchlaufen, um den Merkmalsprozentsatz zu finden, bei dem eine bestimmte Klassifikator-Evaluierungsmetrik ihren größten Wert erreicht. Die beiden in diesem Papier untersuchten Metriken sind die Genauigkeit und das fehlerhafte F-Maß (Aggregat aus fehlerhafter Präzision und Wiedererkennung), obwohl auch andere Metriken weiter untersucht werden könnten. Definitionen der Genauigkeit, des F-Maßes und des ROC können in einem einführenden Text zum maschinellen Lernen nachgelesen werden.

III. EXPERIMENTELLER KONTEXT

Wir haben die Revisionshistorie von Apache, Columba, Gaim, Gforge, Jedit, Mozilla, Eclipse, Plone, PostgreSQL, Subversion und einem kommerziellen, in Java geschriebenen Projekt (JCP) gesammelt.

TABELLE I
ZUSAMMENFASSUNG DER
UNTERSUCHTEN PROJEKTE

Projekt	Zeitraum	Sauber Änderungen	Buggy Änderungen	Eigenschaften
APACHE 1.3	10/1996-01/1997	579	121	17,575
COLUMBA	05/2003-09/2003	1,270	530	17,411
GAIM	08/2000-03/2001	742	451	9,281
GFORGE	01/2003-03/2004	339	334	8,996
JEDIT	08/2002-03/2003	626	377	13,879
MOZILLA	08/2003-08/2004	395	169	13,648
ECLIPSE	10/2001-11/2001	592	67	16,192
PLONE	07/2002-02/2003	457	112	6,127
POSTGRESQL	11/1996-02/1997	853	273	23,247
SUBVERSION	01/2002-03/2002	1,925	288	14,856
JCP	1 Jahr	1,516	403	41,942
Insgesamt	K.A.	9,294	3,125	183,054

Mit Ausnahme von JCP handelt es sich um ausgereifte Open-Source-Projekte. Diese Projekte werden in diesem Dokument gemeinsam als Korpus bezeichnet.

Unter Verwendung der CVS- (Concurrent Versioning System) oder SVN- (Subversion) Quellcode-Repositories des Projekts wurden die Revisionen 500-1000 für jedes Projekt gesammelt, mit Ausnahme von Jedit, Eclipse und JCP. Für Jedit und Eclipse wurden die Revisionen 500-750 gesammelt. Für JCP wurden die Änderungen eines ganzen Jahres gesammelt. Tabelle I gibt einen Überblick über die in dieser Studie untersuchten Projekte und die Dauer der einzelnen Projekte.

IV. ERGEBNISSE

In den folgenden Abschnitten werden die Ergebnisse vorgestellt, die bei der Prüfung der drei Forschungsfragen erzielt wurden.

A. Vergleich der Klassifikatorleistung

Die beiden Hauptvariablen, die sich auf die Leistung der Fehlervorhersage auswirken, werden in dieser Arbeit untersucht: (1) der Typ des Klassifikators (Na¨ıve Bayes, Support Vector Machine) und (2) die Metrik (Genauigkeit, F-Maß), auf die der Klassifikator optimiert ist. Die vier Permutationen dieser Variablen werden in allen 11 Projekten des Datensatzes untersucht. Für SVMs wird ein linearer Kernel mit Standardwerten für den Schlupf verwendet. Für jedes Projekt wird eine Merkmalsauswahl durchgeführt, gefolgt von der Berechnung der Genauigkeit pro Projekt, der Fehlerpräzision, des Fehlerrückrufs und des Fehler-F-Maßes. Sobald alle Projekte abgeschlossen sind, werden die Durchschnittswerte für alle Projekte berechnet. Die Ergebnisse sind in Tabelle II aufgeführt.

B. Auswirkung der Merkmalsauswahl

Im vorangegangenen Abschnitt wurde die durchschnittliche Leistung der verschiedenen Klassifizierer und Optimierungskombinationen über alle Projekte hinweg verglichen. In der Praxis wird die Änderungsklassifizierung für ein bestimmtes Projekt trainiert und eingesetzt. Daher ist es sinnvoll, die Bandbreite der Leistung zu verstehen

TABELLE II
DURCHSCHNITTLICHE LEISTUNG DES KLASSIFIKATORS IM KORPUS

Technik	Eigenschaften Prozentsatz	Genauigkeit	Buggy Präzision	Buggy Rückruf	Buggy F-Maßnahme
Bayes F-Maß	6.83	0.91	0.96	0.67	0.79
SVM F-Maß	7.49	0.81	0.82	0.54	0.62
Bayes-Genauigkeit	6.92	0.86	0.92	0.53	0.65
SVM-Genauigkeit	7.83	0.86	0.96	0.51	0.65

TABELLE III
NA¨IVE BAYES MIT F-MEASURE-OPTIMIERTER MERKMALSAUSWAHL

Projekt Name	Eigenschaften	Genauigkeit	Buggy Präzision	Buggy Rückruf	Buggy F-Maßnahme	Buggy ROC
APACHE	465	0.92	1	0.56	0.72	0.78
COLUMBA	1618	0.9	0.99	0.67	0.8	0.83
ECLIPSE	802	0.98	0.98	0.79	0.88	0.88
GAIM	1065	0.86	0.96	0.65	0.78	0.83
GFORGE	2954	0.83	0.8	0.83	0.82	0.91
JCP	1041	0.95	1	0.77	0.87	0.89
JEDIT	847	0.9	0.99	0.73	0.84	0.88
MOZILLA	496	0.92	1	0.73	0.85	0.87
PLONE	277	0.91	0.98	0.57	0.72	0.84
PSQL	2504	0.87	0.88	0.54	0.67	0.78
SVN	438	0.94	0.96	0.56	0.71	0.78
Durchschnitt	1137	0.91	0.96	0.67	0.79	0.84

durch Änderungsklassifizierung mit einem reduzierten Merkmalssatz erreicht. Tabelle III enthält für jedes Projekt die Gesamtvorhersagegenauigkeit, die Fehlerpräzision, den Rückruf, das F-Maß und die ROC-Fläche unter der Kurve (AUC) für den Na¨ıve-Bayes-Klassifikator mit F-Maß-optimierter Merkmalsauswahl.

Bei der Betrachtung dieser beiden Tabellen fällt auf, dass drei Projekte mit dem Na¨ıve Bayes-Klassifikator eine Buggy-Präzision von 1 erreichen, was bedeutet, dass alle Buggy-Vorhersagen korrekt sind (keine Buggy False Positives). Waehrend die Buggy Recall-Werte (zwischen 0,54 und 0,83, mit einem durchschnittlichen Buggy Recall von
0,69 für Projekte mit einer Genauigkeit von 1) deuten darauf hin, dass nicht alle Fehler vorhergesagt werden, dennoch wird im Durchschnitt mehr als die Hälfte aller Projektfehler erfolgreich vorhergesagt.

Die Abbildungen 1 und 2 fassen die relative Leistung der beiden Klassifikatoren zusammen und vergleichen sie mit der früheren Arbeit von Kim et al. Bei der Betrachtung dieser Abbildungen wird deutlich, dass die Merkmalsauswahl sowohl die Genauigkeit als auch das Buggy-F-Maß der Fehlervorhersage mittels Änderungsklassifikation deutlich verbessert. Da die Genauigkeit oft auf Kosten der Wiederauffindbarkeit erhöht werden kann und umgekehrt, haben wir die Klassifikatoren anhand des F-Maßes für Fehler verglichen. Gute F-Maße weisen auf die Gesamtqualität der Ergebnisse hin.

Die Ergebnisse von Kim et al. in beiden Abbildungen stammen aus [1], wo sie unter Verwendung desselben Korpus (mit Ausnahme von JCP, das nicht im Korpus von Kim et al. enthalten war, und zwei Projekten, bei denen nicht zwischen fehlerhaften und neuen Merkmalen unterschieden wurde) mit einem SVM-Klassifikator berechnet wurden, der auf einem Satz von Merkmalen trainiert wurde, bei dem keine Merkmalsauswahl vorgenommen wurde (d.h. in der Arbeit von Kim et al. wurden wesentlich mehr Merkmale für jedes Projekt verwendet). Tabelle II zeigt die drastische Verringerung der durchschnittlichen Anzahl von Merkmalen pro Projekt bei Verwendung von Klassifikatoren, die auf einem reduzierten Merkmalssatz trainiert wurden, im Vergleich zur früheren Arbeit von Kim et al.

Zu den zusätzlichen Vorteilen des reduzierten Merkmalssatzes gehören eine höhere Klassifizierungsgeschwindigkeit und Skalierbarkeit. Wir haben festgestellt

Abb. 1. Klassifikator-Genauigkeit nach Projekt

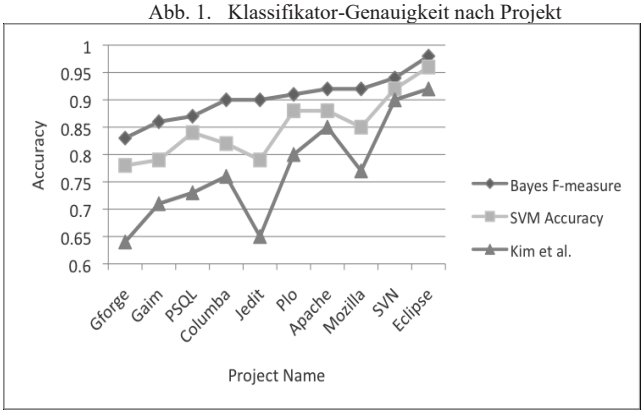
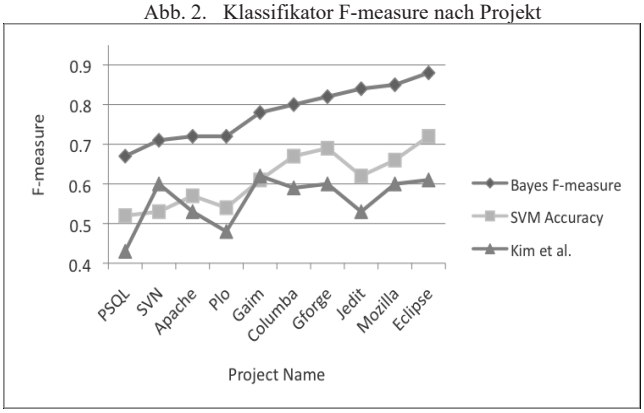


Abb. 2. Klassifikator F-measure nach Projekt



eine Verringerung der Klassifizierungszeit von mehreren Sekunden auf den Bruchteil einer Sekunde in vielen der Projekte. Dies trägt dazu bei, die interaktive Nutzung des Systems in einer integrierten Entwicklungsumgebung zu fördern.

V. VERBUNDENE ARBEITEN

Bei einem Softwareprojekt mit einer Reihe von Programmeinheiten (Dateien, Klassen, Methoden oder Funktionen oder Änderungen je nach Vorhersagetechnik und Sprache) gibt ein Algorithmus zur Fehlervorhersage eines der folgenden Ergebnisse aus.

Vollständig geordnete Programmeinheiten. Eine Gesamtordnung der Programmeinheiten von der höchsten bis zur geringsten Fehleranfälligkeit [10] unter Verwendung einer Ordnungsmetrik wie der vorhergesagten Fehlerdichte für jede Datei [11]. Falls gewünscht, kann dies verwendet werden, um eine Teilordnung zu erstellen (siehe unten).

Teilweise geordnete Programmeinheiten. Eine teilweise Einteilung von Programmeinheiten in fehleranfällige Kategorien (z. B. die *N* % der fehleranfälligsten Dateien in [11]-[13])

Vorhersage für eine bestimmte Software-Einheit. Eine Vorhersage darüber, ob eine bestimmte Softwareeinheit einen Fehler enthält. Die Vorhersagegranularität reicht von einer ganzen Datei oder Klasse [2], [14] bis zu einer einzelnen Änderung (z. B. Change Classification [1]).

A. *Vollständig bestellte Programmeinheiten*

Khoshgoftaar und Allen haben ein Modell zur Auflistung von Modulen nach Software-Qualitätsfaktoren vorgeschlagen, wie z. B. zukünftige

Fehlerdichte mittels schrittweiser Multiregression [10], [15], [16]. Ostrand et al. identifizierten die obersten 20 Prozent der problematischen Dateien in einem Projekt [11] mit Hilfe zukünftiger Fehlerprädiktoren und eines linearen Regressionsmodells.

B. Teilweise bestellte Programmeinheiten

Im vorigen Abschnitt wurden Arbeiten behandelt, die auf einer Gesamtordnung aller Programmmodule beruhen. Dies könnte in eine teilweise geordnete Programmliste umgewandelt werden, z. B. durch Darstellung der obersten *N* % der Module, wie von Ostrand et al. oben ausgeführt. Hassan und Holt verwenden einen Caching-Algorithmus, um die Menge der fehleranfälligen Module zu berechnen, die so genannte Top-10-Liste [13]. Kim et al. schlugen den Bug-Cache-Algorithmus zur Vorhersage künftiger Fehler auf der Grundlage früherer Fehlerlokalisierungen vor [12].

C. Vorhersage für eine bestimmte Softwareeinheit

Unter Verwendung von Entscheidungsbäumen und neuronalen Netzen, die objektorientierte Metriken als Merkmale verwenden, sagten Gyimothy et al. [14] die Fehlerklassen des Mozilla-Projekts über mehrere Versionen hinweg voraus. Ihre Fehlerpräzision und -erinnerung liegen beide bei etwa 70%, was zu einem Fehler-F-Maß von 70% führt. Unsere Fehlerpräzision für das Mozilla-Projekt liegt bei 100% (+30%) und der Recall bei 73% (+3%), was zu einem Fehler-F-Maß von 85% (+15%) führt. Darüber hinaus sagen sie Fehler auf der Klassenebene der Granularität (typischerweise nach Datei) voraus, während unsere Granularität auf der Ebene der Codeänderung liegt, die typischerweise nur 20 Codezeilen umfasst. Aversano et al. [17] erreichten 59 % Fehlerpräzision und -erinnerung, indem sie KNN (K nearest neighbors) zur Lokalisierung fehlerhafter Module verwendeten. Hata et al. [2] zeigten, dass eine Technik, die für die Spam-Filterung von E-Mails verwendet wird, erfolgreich auf Softwaremodule angewendet werden kann, um Software als fehlerhaft oder sauber zu klassifizieren. Sie erreichten eine Genauigkeit von 63,9 %, einen Wiedererkennungswert von 79,8 % und ein F-Maß von 71 % für die besten Datenpunkte der Quellcode-Historie für zwei Eclipse-Plugins. Wir erhalten fehlerhafte Präzisions-, Rückruf- und F-Maßzahlen von 98% (+34,1%), 79% (-0,8%) und 88% (+17%)
bzw. mit unserer besten Technik für das Eclipse-Projekt (Tabelle III). Menzies et al [18] erzielen bei ihren besten Projekten gute Ergebnisse. Im Durchschnitt ist die Genauigkeit jedoch gering und reicht von einem Minimum von 2 % über einen Median von 20 % bis zu einem Maximum von 70 %. Sowohl Menzies als auch Hata konzentrieren sich auf die Dateigranularität. Kim et al. zeigten, dass die Verwendung von Support-Vektor-Maschinen auf Software-Revisionshistorie-Informationen eine durchschnittliche Fehler-Vorhersagegenauigkeit von 78 %, ein Fehler-F-Maß von 60 %, eine Präzision und einen Recall von 60 % liefern kann, wenn sie an zwölf Open-Source-Projekten getestet wurden [1]. Unsere entsprechenden Ergebnisse sind eine Genauigkeit von 91% (+13%), ein Buggy F-measure von 79% (+19%), eine Präzision von 96% (+36%) und eine Recall von 67%.
(+7%).

VI. SCHLUSSFOLGERUNG

In diesem Beitrag wurde die Verwendung eines Algorithmus zur Merkmalsauswahl untersucht, um die Anzahl der Merkmale, die von einem Klassifikator für maschinelles Lernen zur Fehlervorhersage verwendet werden, erheblich zu verringern. Ein wichtiges pragmatisches Ergebnis ist, dass die Merkmalsauswahl in Schritten von 10 % aller Merkmale durchgeführt werden kann, so dass sie schnell erfolgen kann. Zwischen 4,1% und 12,52% der gesamten

Der reduzierte Merkmalsatz führte zu optimalen Klassifizierungsergebnissen. Der reduzierte Merkmalsatz ermöglicht bessere und schnellere Fehlervorhersagen.

Die wichtigsten Ergebnisse der Arbeit finden sich in Tabelle III, in der die für das F-Maß optimierten Ergebnisse für den Na¨ıve-Bayes-Klassifikator dargestellt sind. Erstaunlich ist, dass drei Projekte eine Fehlerpräzision von 1 haben, was bedeutet, dass es bei ihren Fehlervorhersagen keine falsch positiven Ergebnisse gibt. Die durchschnittliche Fehlerpräzision beträgt 0,96, was ebenfalls sehr hoch ist. Aus der Sicht eines Entwicklers, der Fehlervorhersagen zu seiner Arbeit erhält, bedeuten diese Zahlen, dass der Klassifikator fast immer richtig liegt, wenn er sagt, dass eine Änderung einen Fehler enthält.

Wenn Softwareentwickler in Zukunft über eine fortschrittliche Technologie zur Fehlervorhersage verfügen, die in ihre Softwareentwicklungsumgebung integriert ist, wird der Einsatz von Klassifikatoren mit Merkmalsauswahl schnelle, präzise und genaue Fehlervorhersagen ermöglichen. Mit dem weitverbreiteten Einsatz integrierter Fehlervorhersage können Softwareentwickler in Zukunft die Gesamtqualität des Projekts in kürzerer Zeit steigern, indem sie Fehler erkennen, sobald sie auftreten.

REFERENZEN

[1] S. Kim, E. W. Jr., und Y. Zhang, "Classifying Software Changes: Clean or Buggy?" *IEEE Trans. Software Eng.* vol. 34, no. 2, pp. 181-196, 2008.

[2] H. Hata, O. Mizuno, and T. Kikuno, "An Extension of Fault-prone Filtering using Precise Training and a Dynamic Threshold," *Proc. MSR 2008*, 2008.

[3] J. Madhavan und E. Whitehead Jr, "Predicting Buggy Changes Inside an Integrated Development Environment," *Proc. 2007 Eclipse Technology eXchange*, 2007.

[4] A. Mockus und L. Votta, "Identifying Reasons for Software Changes using Historic Databases", *Proc. ICSM 2000*, S. 120, 2000.

[5] J. Sliwerski, T. Zimmermann, und A. Zeller, "When Do Changes Induce Fixes?" *Proc. MSR 2005*, S. 24-28, 2005.

[6] S. T. http://www.scitools.com/, "Maintenance, Understanding, Metrics and Documentation Tools for Ada, C, C++, Java, and Fortran", 2005.

[7] S. Scott und S. Matwin, "Feature Engineering für die Textklassifizierung".
Machine Learning - Internationaler Workshop, S. 379-388, 1999.

[8] E. Alpaydin, *Einführung in das maschinelle Lernen*. MIT Press, 2004.

[9] H. Liu und H. Motoda, *Feature Selection for Knowledge Discovery and Data Mining*. Springer, 1998.

[10] T. Khoshgoftaar und E. Allen, "Predicting the Order of Fault-Prone Modules in Legacy Software," *Proc. 1998 Int'l Symp. on Software Reliability Eng.* , S. 344-353, 1998.

[11] T. Ostrand, E. Weyuker, und R. Bell, "Predicting the Location and Number of Faults in Large Software Systems," *IEEE Trans. Software Eng.* , vol. 31, no. 4, pp. 340-355, 2005.

[12] S. Kim, T. Zimmermann, E. W. Jr. und A. Zeller, "Predicting Faults from Cached History," *Proc. ICSE 2007*, pp. 489-498, 2007.

[13] A. Hassan und R. Holt, "Die Top Ten Liste: Dynamische Fehlervorhersage,"
Proc. ICSM'05, Jan 2005.

[14] T. Gyimo'thy, R. Ferenc und I. Siket, "Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction," *IEEE Trans. Software Eng.* Bd. 31, Nr. 10, S. 897-910, 2005.

[15] T. Khoshgoftaar und E. Allen, "Ordering Fault-Prone Software Modules," *Software Quality J.* , vol. 11, no. 1, pp. 19-37, 2003.

[16] R. Kumar, S. Rai, und J. Trahan, "Neural-Network Techniques for Software-Quality Evaluation," *Reliability and Maintainability Symposium*, 1998.

[17] L. Aversano, L. Cerulo, and C. Del Grosso, "Learning from Bug-introducing Changes to Prevent Fault Prone Code," in *Proceedings of the Foundations of Software Engineering*. ACM New York, NY, USA, 2007, S. 19-26.

[18] T. Menzies, J. Greenwald, und A. Frank, "Data Mining Static Code Attributes to Learn Defect Predictors," *IEEE Trans. Software Eng.* Bd. 33, Nr. 1, S. 2-13, 2007.