Class-Test: Classificação automática de textos para auxiliar a criação de suítes de teste

Leonardo S. Lima, Flávia A. Barros, and Ricardo B. C. Prudêncio

Centro de Informática, Universidade Federal de Pernambuco,
CEP 50732-970, Recife (PE) - Brasil
{lms2,fab,rbcp}@cin.ufpe.br

Resumo. A etapa inicial do processo de teste de software inclui a criação das chamadas *suítes* de teste, que devem conter todos os testes a serem executados, cobrindo vários tipos. O maior problema enfrentado na criação de suítes extensas é o tempo gasto na classificação manual dos testes. Visando diminuir o tempo e o esforço com essa classificação manual, foi construído o Class-Test, um sistema para classificação automática de casos de teste que cobre três tipos de teste (de fronteira, negativo e de interação). Experimentos realizados com quatro algoritmos diferentes revelaram uma taxa de precisão acima de 90% na classificação dos testes. Esta é uma aplicação de classificação de texto inovadora e com grande contribuição para o processo de teste de software.

1 Introdução

Com a crescente demanda por produtos de software de qualidade, torna-se necessário diagnosticar com precisão os problemas que podem ocorrer nas suas várias etapas de desenvolvimento. Nesse contexto, as atividades de testes têm recebido muita atenção das empresas de software, pois podem reduzir os custos e o tempo em torno de 30% a 50% do total do projeto do sistema [1].

Um processo rigoroso de teste de software inclui uma etapa de planejamento, na qual os testes a serem executados são selecionados, categorizados e ordenados seguindo critérios estabelecidos pelos arquitetos de teste. Essa etapa inclui a criação das chamadas *suítes* de teste, que contêm todos os testes a serem executados, cobrindo várias classes (i.e., tipos de teste, tais como testes negativos, de fronteira etc). O maior problema enfrentado na criação de suítes extensas (com mais de 1.000 testes) é o tempo gasto na classificação manual dos testes.

Neste contexto, a classificação automática de testes surge como uma abordagem promissora para diminuir o gargalo dessa etapa inicial do processo de testes. Neste trabalho, lidamos com casos de teste manuais, que são documentos textuais contendo descrições em linguagem natural (e.g., português ou inglês) dos passos a seguir para testar o software em questão.

Visando diminuir o esforço com a classificação dos testes, foi construído o Class-Test [2], para auxiliar a criação de suítes de teste extensas. O sistema, que foi construído com base em técnicas de Aprendizagem de Máquina, conta com três classificadores, um para cada classe alvo (teste de fronteira, negativo

e de interação). Foram criados três classificadores porque alguns casos de teste podem ser associados a mais de uma categoria ao mesmo tempo.

Foram realizados experimentos com um corpus de 879 documentos de testes já etiquetados. Nesses experimentos, diferentes classificadores foram avaliados com validação cruzada 10-fold. O melhor desempenho de classificação foi observado com o uso de Máquinas de Vetores Suporte, que de fato têm mostrado bom desempenho em tarefas de classificação de texto [3]. Os experimentos indicaram ainda taxas de precisão compatíveis com as obtidas na classificação manual, porém com um ganho considerável no tempo de execução dessa tarefa.

Esta pesquisa faz parte de um projeto mais abrangente de automação de testes, o CIn/BTC (Brazil Test Center), desenvolvido por uma parceria entre o CIn-UFPE e a Motorola, com financiamento do CNPq e da Lei de Informática. O presente trabalho é uma aplicação de classificação de texto inovadora, de grande utilidade para diminuir o tempo e o esforço com a classificação de testes manuais.

A seção 2 traz conceitos sobre o processo de teste de software. A seção 3 descreve o Class-Test, seguida da apresentação dos experimentos (seção 4). Por fim, a seção 5 traz a conclusão, indicando trabalhos futuros, alguns já em andamento.

2 Teste de Software

Inicialmente, o teste de software foi definido como uma atividade à parte do desenvolvimento, e que só era realizada ao final do desenvolvimento dos sistemas. Essa visão tradicional não se mostrou eficiente, devido aos altos custos associados a correções dos erros encontrados nos sistemas. Isso contribuiu para a definição de métodos sistemáticos de teste que constituíssem um processo que pudesse ser aplicado em paralelo, ao longo do desenvolvimento do software [4].

O sucesso do teste de software depende de se elaborar um bom planejamento, e seguir todas as fases do ciclo de desenvolvimento do software. Para isso, define-se um processo que inclui atividades de planejamento, análise, codificação e execução dos testes. O trabalho aqui relatado teve como foco automatizar parte da atividade de planejamento dos testes. As seções a seguir tratam dos conceitos de teste de software diretamente relacionados ao nosso trabalho.

2.1 Planejamento dos Testes

A atividade de planejamento consiste em documentar o processo de teste baseado nas expectativas do cliente ou dos stakeholders. Nesta atividade, é necessário definir os recursos necessários à realização dos testes, alocação de pessoal, o escopo e os riscos. O documento onde são registrados todos os passos necessários à execução do teste é conhecido como Plano de Teste [5]. Esse documento deve especificar as abordagens e tipos de teste a serem realizados, os recursos disponíveis, o cronograma das atividades de teste e as funcionalidades cobertas. Abaixo enumeramos alguns dos itens que compõem um Plano de Teste [5].

– Ambiente: informações necessárias para a preparação do ambiente a fim de iniciar a execução dos testes, incluindo a configuração dos itens necessários, como hardware e software;

– Testes: conjunto de casos de teste a serem executados para testar o software (suítes de teste);
– Priorização: definição da ordem de prioridade de execução dos casos de testes definidos no item anterior, com base no fluxo de controle do sistema.

Em geral, as empresas mantêm repositórios de casos de testes (CTs) já desenvolvidos para seus produtos, a fim de reusá-los para testar novos softwares ou para re-testar algum software antigo. Para cada novo plano de testes, os arquitetos devem estipular a quantidade total de testes a serem executados no software em questão, bem como os percentuais de testes de cada tipo (seção 2.3). Aqui, esses profissionais enfrentam dois problemas: (1) seleção inicial dos testes que irão compor os planos, uma vez que esses repositórios tendem a guardar centenas (ou até milhares) de CTs; e (2) classificação manual dos testes pré-selecionados, a fim de obedecer os percentuais estipulados anteriormente.

O problema (1) é, em geral, resolvido com heurísticas, como a escolha de CTs mais recentes, ou que registrem uma maior taxa de falhas em softwares nos quais já foram aplicados. O problema (2) pode ser tratado via classificação automática de texto, principalmente quando o número de testes selecionados chega à ordem das centenas. A priorização dos testes também pode ser automatizada, baseando-se em atributos dos testes versus características do ambiente e do software sendo testado. Contudo, claramente não se trata de uma tarefa de classificação.

2.2 Abordagens e Tipos de Teste

As abordagens de teste mais conhecidas na literatura são o Teste Estrutural (ou *caixa branca*) e Teste Funcional (ou *caixa preta*) [6]. Outras abordagens mais recentes são a Híbrida (ou *caixa-cinza*), e o teste baseado em falhas. Nos testes estruturais, são verificados os fluxos de caminho, de controle e as informações dos fluxos de dados, que dependem da implementação do sistema. Já na abordagem Funcional, casos de testes independentes da implementação são escritos com base na especificações do sistema, a fim de se verificar se o código está funcionando corretamente. Deste modo, se a implementação do sistema for alterada, os mesmos casos de teste continuam sendo úteis. O trabalho de pesquisa aqui relatado desenvolveu-se dentro desta abordagem de teste.

Veremos agora alguns tipos de teste organizacionais. Cada tipo de teste busca detectar um problema diferente no software. São eles: teste de fronteira, negativo, teste de interação entre funcionalidades, teste de stress, de performance, de carga, e por fim, primary functionality e second functionality. Detalharemos aqui apenas os três tipos de teste que foram alvo deste trabalho.

– Teste de Fronteira define valores de fronteira de cada variável do software. Ao invés de utilizar um valor aleatório, este tipo de teste verifica os limites superiores e/ou inferiores das variáveis do sistema;
– Teste de Interação é utilizado quando há uma necessidade de verificar a comunicação entre funcionalidades presentes no software. As funcionalidades são testadas separadamente, depois são integradas para se verificar se estão interagindo corretamente entre si;

– Teste Negativo define uma condição ou informação inaceitável, inválida, ou inesperada, a fim de garantir que os fluxos do sistema só sejam executados quando as condições estiverem corretas. O intuito do caso de teste negativo é ter um comportamento contrário ao daqueles testes que tentam verificar se a funcionalidade se comporta de acordo com o que foi especificado.

3 Class-Test

Examinando o processo de criação dos planos de teste, verificou-se a possibilidade de automatizar algumas de suas etapas. Escolhemos automatizar a classificação dos testes, uma vez que ela consome muito tempo, devido ao grande número de testes envolvidos na formação de suítes extensas. Dentro desse escopo, escolhemos abordar inicialmente Casos de Testes (CTs) do tipo Fronteira, Negativo e de Interação, por serem muito comuns, além de serem indispensáveis em qualquer plano de teste. Além disso, esses casos de teste apresentam regularidades que podem ser capturadas por algoritmos de aprendizagem automática.

O Class-Test foi concebido utilizando técnicas de Aprendizagem de Máquina para classificação de texto. Segundo [3], a classificação automática de texto pode ser usada para organizar documentos em categorias previamente determinadas, facilitando o acesso à informação relevante. O sistema conta com três classificadores binários (um para cada categoria). Cada classificador responde se um CT de entrada pertence ou não a uma dada categoria (valor de classe *positiva* e *negativa*, respectivamente). Essa abordagem equivale à técnica de *Relevância Binária* [7], usada em problemas de classificação *multi-label* (múltiplos rótulos). Veremos a seguir as etapas de criação dos classificadores para o Class-Test.

3.1 Aquisição e Pré-processamento dos Documentos

Aquisição - O projeto CIn-BTC trabalha com um repositório digital de casos de teste que armazena milhares de testes de vários tipos. Como já dito, nossos CTs trazem a descrição, em linguagem natural, dos passos que o testador deve seguir para testar o software em questão. O corpus usado para construção dos classificadores foi coletado a partir desse repositório, contando com 879 CTs de diferentes componentes. Os dados foram manualmente etiquetados com a ajuda de especialistas no domínio. A distribuição dos 879 documentos em cada categoria foi: (1) Fronteira (196 positivos e 683 negativos); (2) Negativo (354 positivos e 525 negativos); e (3) Interação (364 positivos e 515 negativos).
Pré-processamento dos documentos - Cada documento foi representado inicialmente na forma de uma lista de palavras (ou termos). Esses termos foram oriundos dos campos mais relevantes dos CTs: *Descrição*, *Condições Iniciais*, *Passos* e *Resultados Esperados*. Esses campos foram extraídos de forma automática, considerando oito padrões de CTs encontrados no corpus. Contudo, é possível que existam, no repositório de testes, CTs com outros padrões que não foram cobertos. Para resolver este problema, será construído um software capaz de lidar com qualquer padrão de CT usando técnicas de Extração de Informação.

As listas de termos representando os CTs foram reduzidas pela eliminação de *stopwords* (termos sem relevância, como artigos, preposições e conjunções). A seguir, utilizou-se o algoritmo Porter para realizar a operação de *stemming*, que visa reduzir cada palavra à sua (provável) raiz. Como resultado, cada CT passou a ser representado por uma lista de *stems*, e não mais de palavras completas.

3.2 Criação do Vocabulário do Corpus e Seleção de Atributos

Para criar os conjuntos de treinamento, os documentos devem ter uma representação final uniformizada com base em um vocabulário. No nosso trabalho, dado um vocabulário de T termos, cada CT é descrito por um vetor de T características, onde cada característica é um valor booleano que indica a presença ou não de um dado termo no CT. A escolha da representação booleana se deve fato de que os CTs, em geral, são curtos e cada *stem* é observado comumente apenas uma vez por documento. Assim, acreditamos que esquemas de representação mais sofisticados (e.g., TF-IDF [8]) não influenciariam muito o desempenho dos classificadores (embora esse aspecto deva ser melhor investigado no futuro).

Inicialmente, criamos um vocabulário pela união de todas as listas de stems dos CTs, o que resultou em um vocabulário com 2.789 termos. Esse vocabulário foi reduzido, através de uma operação de seleção de atributos, que levou em consideração a frequência de ocorrência de cada termo no corpus. A seleção de atributos foi feita separadamente por categoria de CTs (ou seja, por conjunto de treinamento). Para cada categoria, foi definido um vocabulário com os 700 termos de maior ocorrência nos CTs da categoria. Os vocabulários podem variar (ainda que pouco) de categoria para categoria, uma vez que um termo pode ser relevante para o aprendizado de uma categoria e não ser relevante para outra.

3.3 Indução dos Classificadores

Para cada um dos três problemas de classificação, foram criados classificadores usando quatro algoritmos implementados no ambiente WEKA [9]: (1) o SMO (SVMs treinadas com Sequential Minimal Optimization [10]); (2) o Naive Bayes (NB) [11]; (3) o IBk (implementação de k-Vizinhos mais Próximos [12]); e (4) o J48 (versão do algoritmo C4.5 para árvores de decisão [13]). Todos os algoritmos foram executados com os parâmetros default do WEKA.

4 Experimentos e Resultados

No nosso trabalho, cada algoritmo considerado na seção 3.3 foi avaliado usando validação cruzada 10-fold e as taxas de precisão assim como outras medidas de avaliação foram comparadas. A tabela 1 apresenta os resultados de cada algoritmo, considerando a categoria Fronteira. O algoritmo SMO apresentou a melhor precisão, seguido por J48 e IBk, que também mostraram desempenho satisfatório. Ressaltamos que esse conjunto de exemplos é mais fortemente desbalanceado que os demais, o que leva os classificadores a responderem melhor

para a classe majoritária. Nesse aspecto, SMO apresentou boa precisão e cobertura também para a classe positiva. O IBk e J48, por sua vez, apresentaram uma diferença de cobertura para as classes positiva e negativa em torno de 10%.

Tabela 1. Desempenho dos algoritmos para a categoria Fronteira.

Algoritmo	Precisão Global	Classe	No. de Exemplos	Precisão	Cobertura
J48	0.960	Positiva	196	0.951	0.893
		Negativa	683	0.970	0.987
NB	0.921	Positiva	196	0.895	0.781
		Negativa	683	0.939	0.974
SMO	0.976	Positiva	196	0.979	0.944
		Negativa	683	0.984	0.994
IBk	0.950	Positiva	196	0.915	0.878
		Negativa	683	0.965	0.977

A tabela 2 apresenta o desempenho dos algoritmos para a categoria Interação. Aqui também o algoritmo SMO obteve a melhor taxa de precisão. O IBk foi o algoritmo que chegou mais próximo do SMO, enquanto o J48 e NB tiveram desempenho mais baixo. Ao contrário do que ocorreu em Fronteira, nesse classificador, a diferença de desempenho obtido pelos melhores algoritmos (i.e., SMO e IBk) para as classes majoritária e minoritária não é tão grande. Isso pode ter ocorrido pelo fato do conjunto de dados para o classificador Interação estar mais balanceado. O resultado do experimento com o classificador Negativo pode ser visto na tabela 3. Neste caso, o algoritmo SMO não apresentou a melhor taxa de precisão, tendo sido superado pelo IBk, contudo com uma pequena diferença de precisão. Os algoritmos J48 e NB apresentaram os piores resultados.

Tabela 2. Desempenho dos algoritmos para a categoria Interação.

Algoritmo	Precisão Global	Classe	No. de Exemplos	Precisão	Cobertura
J48	0.918	Positiva	364	0.915	0.893
		Negativa	515	0.926	0.942
NB	0.916	Positiva	364	0.866	0.962
		Negativa	515	0.971	0.895
SMO	0.971	Positiva	364	0.959	0.959
		Negativa	515	0.976	0.971
IBk	0.958	Positiva	364	0.944	0.975
		Negativa	515	0.982	0.959

Tabela 3. Desempenho dos algoritmos para a categoria Negativo.

Algoritmo	Precisão Global	Classe	No. de Exemplos	Precisão	Cobertura
J48	0.912	Positiva	354	0.913	0.887
		Negativa	525	0.925	0.943
NB	0.909	Positiva	354	0.945	0.822
		Negativa	525	0.890	0.968
SMO	0.947	Positiva	354	0.928	0.946
		Negativa	525	0.963	0.950
IBk	0.954	Positiva	354	0.934	0.958
		Negativa	525	0.971	0.954

Realizamos ainda um experimento de classificação manual com dois arquitetos de teste. Cada arquiteto realizou a classificação de 60 CTs (20 por categoria), selecionados aleatoriamente dos exemplos disponíveis. A precisão obtida pelos dois humanos na classificação da amostra foi de 98% e 90% respectivamente, o que é compatível com as taxas de acerto alcançadas na classificação automática. O tempo gasto durante a classificação manual, no entanto, foi em torno de 25 minutos no total. O tempo gasto na classificação automática, considerando o tempo de processamento do texto dos CTs e da classificação em si, foi de apenas 2 minutos. Esse experimento indica que a classificação automática pode alcançar precisão similar a dos especialistas humanos, com um ganho significativo do tempo gasto na classificação. No entanto, esse resultado ainda não é conclusivo, devido ao número reduzido de humanos envolvidos na comparação.

5 Conclusões

Apresentamos aqui o Class-Test, um sistema para classificação automática de casos de testes em categorias pré-definidas, com o objetivo de auxiliar na criação de suítes de teste extensas. Adotamos a abordagem de aprendizagem de máquina, que se mostrou adequada uma vez que os documentos tratados apresentam regularidades passíveis de serem capturadas por algoritmos de aprendizagem. Destacamos aqui a originalidade deste trabalho, no que concerne à área de aplicação. Utilizamos aqui aprendizagem de máquina para tratar o problema de classificação de casos de testes em um ambiente empresarial.

Os experimentos revelaram boas taxas de precisão e cobertura na classificação automática, comparáveis às obtidas por especialistas na classificação manual. Dentre os quatro algoritmos de aprendizagem avaliados, SVMs apresentaram de uma forma geral os melhores desempenhos. Ressaltamos que novos experimentos serão realizados no futuro, em especial para fazer uma comparação mais confiável entre a classificação automática e a classificação manual.

Podemos vislumbrar diversos trabalhos futuros que estendem o trabalho realizado. No nosso trabalho, o processamento dos documentos foi realizado uti-

lizando procedimentos padrões. Assim, vários aspectos podem ser investigados com maior profundidade como o uso de outras técnicas para seleção de atributos, e outros esquemas de representação dos documentos. Pretendemos investigar ainda técnicas de balanceamento de dados, visando aumentar o desempenho dos classificadores para as classes minoritárias. Outras abordagens de classificação multi-label podem ser adotadas ainda em trabalhos futuros. Finalmente, citamos ainda a ampliação das categorias cobertas pelo Class-Test para tratar os tipos de teste de Stress, Performance e Carga, atividade já em andamento.

Referências

1. Peters, J. (2001). Engenharia de Software. Campus, Rio de Janeiro.
2. Lima, L. S. (2009). Class-Test: Classificação automática de testes para auxílio à criação de suítes de teste. *Dissertação de Mestrado*, Centro de Informática, UFPE.
3. Sebastiani, F. (2002). Machine learning in automated text categorization. *ACM Computing Surveys*, 34:1–47.
4. McGregor, J. D., Sykes, D. A. (2001). A Practical Guide to Testing Object-Oriented Software. Addison-Wesley.
5. IEEE (1998). IEEE Std 829: Standard for Software Test Documentation. IEEE Computer Society, New York.
6. Jorgensen, P. C. (2002). Software Testing: A Craftsman's Approach. 2 ed. CRC Press.
7. Tsoumakas, G., Katakis, I. (2007). Multi-label classification: An overview. *International Journal of Data Warehousing and Mining*, 3(3):1-13.
8. Salton, G., Wong, A., Yang, C. (1975). A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620.
9. Witten, I.H., Frank, E., editors (2003). *WEKA: machine learning algorithms in Java*. University of Waikato, New Zealand.
10. Platt, J. (1998) Fast Training of Support Vector Machines using Sequential Minimal Optimization. In *Advances in Kernel Methods - Support Vector Learning*, MIT Press.
11. John, G. H., Langley, P. (1995) Estimating continuous distributions in bayesian classifiers. In *11th Conf. on Uncertainty in Artificial Intelligence*, pages 338–345.
12. Aha, D., Kibler, D. (1991). Instance-based learning algorithms. *Machine Learning*, 6(3):37–66.
13. Quinlan, R. (1993). C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo, CA.