Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação

SÍLVIO ROGÉRIO TASSINI BORGES

APRENDIZADO ATIVO PARA DESCOBERTA DE FALHAS EM CÓDIGOS FONTE UTILIZANDO O PROBLEMA DA MAXIMIZAÇÃO DE DIVERSIDADE

Belo Horizonte
2011

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação
Especialização em Informática: Ênfase: Engenharia de Software

APRENDIZADO ATIVO PARA DESCOBERTA DE FALHAS EM CÓDIGOS FONTE UTILIZANDO O PROBLEMA DA MAXIMIZAÇÃO DE DIVERSIDADE

por

SÍLVIO ROGÉRIO TASSINI BORGES

Monografia de Final de Curso

Profa. Dra. Gisele Lobo Pappa
Orientador(a)

Belo Horizonte
2011

SÍLVIO ROGÉRIO TASSINI BORGES

APRENDIZADO ATIVO PARA DESCOBERTA DE FALHAS EM CÓDIGOS FONTE UTILIZANDO O PROBLEMA DA MAXIMIZAÇÃO DE DIVERSIDADE

Monografia apresentada ao Curso de Especialização em Informática do Departamento de Ciências Exatas da Universidade Federal de Minas Gerais, como requisito parcial para obtenção do certificado de especialista em informática.

Área de concentração: Engenharia de Software

Orientador(a): Profa. Dra. Gisele Lobo Pappa

Belo Horizonte
2011

Aos professores,
aos colegas de curso e
aos meus familiares,
dedico este trabalho.

AGRADECIMENTOS

Agradeço a minha orientadora, sempre paciente e gentil.

Agradeço aos meus pais, por todo apoio.

Agradeço especialmente a Isamara, sempre me incentivando.

E finalmente aos colegas de curso pela convivência e trocas.

“A fórmula da minha felicidade:
um sim, um não,
uma linha reta, um objetivo."
Friedrich Nietzsche

RESUMO

O presente trabalho tem como principal objetivo apresentar uma nova abordagem de Aprendizado Ativo, e propor uma forama de utilizá-lo na busca por código-fontes passível de falha. A tecnologia crescente tem permitido sistemas de software cada vez mais complexos e, por conseguinte, códigos-fonte cada vez maiores. A busca por código com falha torna-se, então, tarefa de alto custo e, por vezes, humanamente inviável. Dessa forma, o problema de busca por falha em código-fonte é melhor tratado quando modelado como um problema de aprendizado. Ainda assim, existe a necessidade de um número de dados, códigos, rotulados para o aprendizado do algoritmo. Para diminuirmos essa necessidade, utilizamos o paradigma do Aprendizado Ativo, que seleciona dentro de um conjunto, um subconjunto de elementos que forneçam informações relevantes para aprendizado. Nesse trabalho determinamos que a seleção seja baseada no problema da diversidade máxima, um problema de otimização que busca selecionar elementos que apresentem maior diversidade em relação a uma característica dentro de um conjunto.

Palavras-chave: Aprendizado Ativo, Problema da Diversidade Máxima, *GRASP*, falhas de software

LISTA DE FIGURAS

LISTA DE SIGLAS

ES	Engenharia de Software
MD	Mineração de Dados
GRASP	Greedy Randomized Adaptive Search Procedure
PMD	Problema da Maximização de Diversidade

SUMÁRIO

1 INTRODUÇÃO

O crescente avanço da Engenharia de Software (ES) é proporcional ao aumento da complexidade dos processos de desenvolvimento de software e do número de artefatos gerados por esses processos. Como conseqüência, uma grande quantidade de dados é gerada durante o processo de desenvolvimento de software, o que inviabiliza uma análise adequada dos mesmos sem o uso de técnicas e apoio computacional (HAN, KAMBER, 2000).

Deste modo, outra área também em expansão em vários campos da ciência da computação vem recebendo atenção na Engenharia de Software: Mineração de dados (MD). A descoberta de conhecimento confiável e instrutivo de uma fonte muito grande de dados torna-se então crucial em várias fases do processo de desenvolvimento de software.

A mineração de dados oferece técnicas e métodos para extração de informação por meio de reconhecimento de padrões ou outra relação pré-determinada. Nesse contexto, o uso de MD mostra-se útil para a ES em diversos momentos, seja na busca de informação dos artefatos gerados durante o processo, seja nos testes ou na análise do código do software.

A Engenharia de Software, dentro do processo de desenvolvimento, armazena códigos-fonte, histórico de execuções, histórico de mudanças, bases de dados de erros, dentre outras informações associadas a criação do sistema. Esse conjunto de dados prove uma fonte de informações acerca do *status* e progresso do projeto, assim como sua evolução. Usando técnicas de mineração de dados adequadas os profissionais envolvidos em projetos de software podem explorar todos esses dados, a fim de gerenciar melhor seus projetos e produzirem software de alta qualidade (PRASAD, KRISHNA, 2010)

Podem-se destacar alguns exemplos de uso de mineração de dados na engenharia de software: (i) *compreensão do sistema*, uma vez que profissionais que participaram do desenvolvimento podem não pertencer a empresa ou não possuírem tempo hábil para ajudar aos novatos; (ii) predição e identificação de erros, em vez de buscar pelos futuros erros o uso de MD pode encontrar padrões em um código com erro e, assim, marcar outros trechos do código com mesmo padrão; (iii) reutilização de código, já que muitas

bibliotecas são complexas de serem usadas, e uma busca no código por padrões de uso pode ajudar um novato a utilizá-la.

A proposta do presente trabalho é encontrar no repositório de códigos do sistema em desenvolvimento partes de código passível de falha. Surge, então, outro problema: como usar MD se há pouco ou nenhum dado rotulado (como falha ou não falha) que determine um padrão? No desenvolvimento de sistemas, mesmo os de pequeno porte, o número de linhas de código pode ser de tal grandeza que, o custo para realizar a tarefa de procura e rotulação de dados torna-se inviável. A Figura 1 ilustra o problema da escolha de quais linhas de código rotular.

Código Fonte	Passível de falha?
Linha de código 1	S
Linha de código 2	N
Linha de código 3	
Linha de código n	

Quais outras linhas rotular?

Figura 1. Código Fonte que necessita de rotulação
Fonte: Próprio autor

Dentro da área do Aprendizado de Máquina existem algoritmos que utilizam o paradigma de Aprendizado Semi-supervisionado. Estes algoritmos necessitam de poucos dados rotulados para aprender a classificar os restantes. Dentro do Aprendizado Semi-supervisionado, um paradigma interessante é o Aprendizado Ativo. O Aprendizado Ativo tem como principal objetivo selecionar um subconjunto de dados relevantes, para que esses sejam manualmente rotulados pelo usuário e depois utilizados para aprender um modelo de classificação.

Para usarmos este pardigma modelaremos nosso problema como um Problema de Maximização de Diversidade (PMD), ou seja, diante de um determinado conjunto desejamos identificar subconjuntos que apresentem características o mais divergentes possíveis e, com esse subconjunto, adquirir conhecimento que possibilite inferências sobre outros subconjuntos analisados posteriormente. Assim, o PMD para Aprendizado Ativo será modelado usando o método heurístico *GRASP* (*Greedy Randomized Adaptive Search Procedure*).

Este trabalho está organizado da seguinte forma. No capítulo 2 apresenta-se a fundamentação teórica do trabalho, os conceitos de Mineração de Dados, da heurística *GRASP* e do Problema de Maximização de Diversidade. No capítulo 3 é apresentado o problema proposto e o caminho para sua solução. No capítulo 4 apresentamos aplicações potenciais da solução no campo de Engenharia de Software.

1.1 OBJETIVO GERAL

Apresentar uma abordagem para seleção de exemplos para aprendizado no contexto da Engenharia de Software utilizando Aprendizado Ativo, modelando o problema através da maximização da diversidade para otimizar a busca por falhas em código fonte.

1.2 OBJETIVO ESPECÍFICO

- Pesquisar e apresentar fundamentação teórica da Mineração de Dados e seu uso na Engenharia de Software.
- Pesquisar e apresentar fundamentação teórica para utilização de metaheurísticas na solução do problema proposto.
- Propor uma abordagem para a solução do problema de maximização de diversidade usando a metaheuristica *GRASP* com Aprendizado Ativo semi-supervisionado.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 MINERAÇÃO DE DADOS

A última década, em especial, foi marcada por grandes transformações no que diz respeito à gestão da informação. A quantidade de informação produzida e vinculada nos diversos meios existentes vem aumentando de forma exponencial, crescimento que ocasiona um problema de relevante importância: Como obter o dado desejado em meio a essa avalanche de informações?

A fim de se extrair o dado requerido e assim obter uma informação de valor, útil a determinado fim, pesquisas em diversos campos do conhecimento se intensificaram e convergiram para o que nos dias atuais tornou-se um campo de pesquisa em crescente aceleração, a Mineração de Dados.

A Mineração de Dados consiste em um processo, uma etapa da descoberta de conhecimento em banco de dados, responsável pela mineração, ou seja, em um conjunto de dados apresentados, analisar padrões e freqüências. As tarefas de Mineração de Dados podem ser classificadas em dois tipos: Descrição de Dados e Predição de Dados (HAN, KAMBER, 2000).

A descrição de dados consiste em apresentar os dados de forma mais clara para que estes possam ser interpretados. Esta tarefa pode ser vista como uma caracterização da classe de determinada informação do conjunto, onde os dados podem ser classificados, separados em grupos por similaridade, analisados de acordo com alguma associação dentre outras opções; já o segundo tipo, a predição de dados, busca padrões no conjunto de informações de modo que seja possível fazer inferências sobre outro conjunto de informações e classificá-lo.

As técnicas usadas para realizar uma determinada tarefa em Mineração de Dados variam de acordo com os padrões buscados e os resultados que se deseja encontrar no conjunto de informações. Existem diversas técnicas para as mais variadas tarefas: técnicas estatísticas (classificadores *Bayseanos*), de aprendizado de máquina (indução por árvores de decisão), análise de *Outliers*, regressão linear, classificação por *Backpropagation*, algoritmos genéticos e outros. Para tarefas de predição e classificação é comum o uso de técnicas que envolvam Algoritmos Genéticos, Redes Neurais, , etc.

Aprendizado de máquina é uma área de pesquisa de métodos computacionais que está relacionada com a capacidade de uma máquina aprender a mudar sua estrutura, programa ou dados de acordo com entradas externas ou respostas a informações externas de modo que seu desempenho seja melhorado. Existe um paralelo entre aprendizado de máquina e a aprendizagem animal e, graças aos avanços das pesquisas no campo da psicologia na área da aprendizagem animal e humana, atualmente muitas técnicas de aprendizagem de máquina são derivadas desses estudos (NILSSON, 1998).

O primeiro passo para o aprendizado da máquina é a modelagem do problema real em termos computacionais, ou seja, definir atributos para objetos (exemplos) do conjunto a ser estudado, de modo que as relações dos atributos dos objetos possam ser analisados por um algoritmo de aprendizado de máquina. Comumente o conjunto de dados a ser analisado é dividido em pelo menos dois outros conjuntos: um conjunto de dados de treinamento, que servirá de entrada para o algoritmo de aprendizado e sobre o qual será realizado o aprendizado de padrões nos dados; um conjunto de testes, que será utilizado para avaliar o modelo criado como resultado do primeiro conjunto.

Existem vários tipos de algoritmos usados para aprendizagem de máquina. Eles podem variar de acordo com a origem do estudo e do modelo: estatístico, cerebral, teoria do controle adaptativo ou inteligência artificial.

Existem três abordagens comumente utilizadas por algoritmos de aprendizado de acordo com o que se conhece acerca de um exemplo. Considere, por exemplo, a tradicional tarefa de classificação. Suponha que queremos classificar códigos-fonte em dois grupos: códigos com falhas ou códigos sem falhas. Se for conhecida uma quantidade expressiva de códigos que pertencem a cada um dos grupos então o aprendizado supervisionado apresentará melhor resultado. Caso não se conheça o grupo de nenhum dos códigos apresentados, então o aprendizado não-supervisionado deverá ser empregado. Por fim, se existe informação do grupo a que pertence um número pequeno de códigos em relação ao montante total de códigos apresentados, o tipo de aprendizagem mais recomendada é a semi-supervisionada.

Como mencionado, os algoritmos de aprendizado semi-supervisionados tratam de problemas onde o número de informações rotuladas é mínimo perto do volume de informações a se analisar. Deste modo, um problema da Mineração de Dados, que trabalha com grande volume de informação, pode ser tratado, uma vez que a rotulação de dados não precisa ser tão extensa. Um humano realiza parte da tarefa e em seguida o restante do conjunto é analisado pelo algoritmo.

Esse trabalho focará em métodos de aprendizado semi-supervisionado, uma vez que eles são os menos explorados na literatura.

2.2 APRENDIZADO ATIVO

Os algoritmos de aprendizado de máquina Semi-supervisionados podem ser entendidos como sendo aqueles que operam a fim de aumentar o número de elementos rotulados e, assim, permitir o uso dos algoritmos Supervisionados. A literatura já apresenta alguns algoritmos de aprendizado semi-supervisionado, dentre os quais destacamos o COP-*k-means* (WAGSTAFF et al, 2001), SEEDED-k-means (BASU, BANERJEE, MOONEY, 2002) e CONSTRAINED-*k-means* (BASU, BANERJEE, MOONEY, 2002) todos utilizando uma variação do algoritmo de aprendizado não supervisionado *k-means*; Outros algoritmos de aprendizado não supervisionado como *Expectation Maximization* e *Support Vector Machine* também foram modificados a fim de se obter algoritmos semi supervisionados. Em comum, estes algoritmos tentam ganhar alguma informação sobre as informações rotuladas e, para isso, a escolha das características analisadas torna-se essencial para o sucesso do algoritmo.

Dentre os vários tipos de algoritmos que seguem o aprendizado semi-supervisionado, estão os algoritmos de Aprendizado Ativo. Esses algoritmos analisam um subconjunto não rotulado de dados e tentam identificar quais exemplos trarão maior ganho de informação ao algoritmo de aprendizado se rotulados.

Existem diversas estratégias que podem ser seguidas, uma delas propõe selecionar o exemplo do conjunto de treinamento e consultar um oráculo, uma função ou mesmo um ser humano, acerca da rotulação do exemplo selecionado. De acordo com o resultado obtido, ou seja, como o exemplo foi rotulado, o algoritmo aprende com a nova informação e busca selecionar novos exemplos. Para medir a importância da informação que o exemplo fornece, existem também algumas estratégias como Consulta a um Comitê e Amostragem por Incerteza. No primeiro caso, o exemplo selecionado é avaliado por um conjunto de oráculos que votam na rotulação ou não do dado, o dado com maior informação potencial será aquele que os oráculos mais divergiram acerca da rotulação. A segunda estratégia é muito simples, e o algoritmo busca o exemplo que ele tem menos certeza sobre seu valor de rótulo (SETTLES, 2009).

Como consequencia, o Aprendizado Ativo diminui o esforço humano na rotulação de dados.

2.3 PROBLEMA DA MAXIMIZAÇÃO DE DIVERSIDADE

O Problema da Maximização da Diversidade (PMD) é um problema da área de otimização combinatória classificado como NP-Difícil; O problema consiste em identificar, dentro de um conjunto, um subconjunto de elementos tal que as características desses elementos apresentem a maior diversidade possível. O Problema da Diversidade Máxima tem aplicações em diversas áreas, tais como: Geolocalização, Ecologia, Tratamentos médicos, Genética, Administração de recursos humanos, dentre outros.

O objetivo do problema é obter um conjunto onde a distância entre cada elemento seja maximizada. A definição de distância é estabelecida de acordo com a natureza do conjunto trabalhado, os atributos dos elementos e da aplicação do problema. A formulação matemática que representará os atributos dos elementos e, por conseguinte, determinará o grau de diferença entre esses elementos é, portanto, fator chave para o sucesso na solução do problema. O modelo matemático fornecerá um valor, chamado grau de diferença, que será usado para selecionar os elementos para o subconjunto.

De acordo com a definição acima, pode-se perceber que a ideia do PMD vai de encontro com os objetivos do Aprendizado Ativo: selecionar um exemplo com as características mais distintas dentro de um conjunto de exemplos pré-definidos. Assim, nesse trabalho propõe-se uma abordagem do problema PDM usando a metaheurística *GRASP*. O foco em Engenharia de Software para aplicação do problema será nos códigos fonte gerados. Estamos interessados em analisar os códigos que podem apresentar falhas, erros. Para tanto, do conjunto de códigos apresentados, será preciso conhecer um número mínimo dos quais apresentam ou não falhas, rotulá-los e, com estes, treinar o algoritmo *GRASP* para resolver o PMD para identificar outros códigos a fim de se obter um subconjunto que possa ser rotulado.

SILVA, OCHI e MARTINS (2006) destacam alguns autores que já propuseram uma heurística para a solução do problema PMD. Entre os trabalhos citados pela autora distinguem-se os estudos de GHOSH (1996) e ANDRADE et al (2003) que propõem a metaheurística *GRASP*. Os dois trabalhos apresentam em comum a idéia de que para alcançar a solução completa do PDM é preciso *m* soluções parciais.

O algoritmo proposto por GHOSH (1996) para o PMD foi testado em pequenos exemplos, mas o resultado obtido comparado ao resultado gerado por um algoritmo exato mostrou-se eficiente alcançando uma solução ótima ou quase ótima (SILVA, OCHI, MARTINS, 2006).

2.4 GRASP

Para obtenção de padrões de um conjunto de informações e posterior predição de comportamento de outros conjuntos, técnicas de aprendizado de máquina envolvidas com MD precisam de um subconjunto de informações previamente rotuladas, isto é, classificadas. Estes elementos rotulados são usados para treinar os algoritmos de aprendizado. É fácil perceber que, como dito inicialmente, a quantidade de informação produzida e vinculada é tamanha que a classificação de um subconjunto pode ser tarefa tão árdua quanto à classificação do conjunto inteiro. Além disso, em aplicações reais, não é comum encontrar informações rotuladas. Portanto, é preciso ter em mente que informações rotuladas são escassas ou podem não existir, dificultando assim o trabalho de aprendizagem da máquina.

A metaheurística *GRASP* é um método comumente usado para resolução de problemas de otimização combinatória, onde se possui um conjunto de dados e, a partir de uma série de regras, pretende-se selecionar subconjuntos que atendam a essas regras a um determinado custo computacional. *GRASP* utiliza características dos algoritmos gulosos e procedimentos aleatórios para a construção de uma solução viável, e funciona em duas fases: (i) construção de uma solução inicial; (ii) busca, onde o método tenta melhorar a qualidade da solução existente procurando por uma solução ótima na vizinhança da solução inicial (FEO, RESENDE, 1995).

Na primeira fase, a de construção, uma solução inicial é construída de forma iterativa, pegando os elementos um a um. A cada iteração dessa fase, os elementos seguintes a serem incluídos na solução são colocados em uma lista de candidatos *C*, seguindo um critério predeterminado. Esse processo de seleção é baseado em uma função adaptativa gulosa f: $C \rightarrow R$, que é responsável por avaliar os benefícios da escolha do elemento. A escolha de um elemento em uma iteração interfere na seleção do elemento posterior, daí o porquê do algoritmo ser dito adaptativo. No conjunto dos

elementos iniciais existe um grupo que se sobressai de acordo com a função *f*, esse subconjunto é chamado de Lista Restrita de Candidatos (LCR). A Figura 2 mostra um pseudocódigo para o algoritmo GRASP.

```
Proc GRASP (max_iterations, seed)
      ler_entrada ()
      para i=1 ate max_iterations faca
            Solucao = constroiSolucao (seed)
            Solucao = buscaLocal ()
            atualizaSolucao (Solucao, Melhor_Solucao)
      fim;
      retorna Melhor_Solucao
fim GRASP
```

Figura 2: Pseudocódigo da metaheurística GRASP.
Fonte: SILVA, OCHI, MARTINS, 2006

Por tratar-se de uma técnica determinística, a solução inicialmente criada pode não ser uma solução ótima. Por isso, o algoritmo analisa a vizinhança da solução inicial buscando melhorar a solução construída. A fase de construção da solução também é iterativa, e construída elemento por elemento, como visto na Figura 3.

```
Proc constroiSolucao (seed)
      Solução = {}
      Avaliar o custo de incrementar elementos
      enquanto solução não for solução completa faca
            construa lista restrita de candidatos (LRC)
            selecione aleatoriamente um elemento e da lista LRC
            Solução = Solução U {e}
            Reavaliar o custo de incrementar elementos
      Fim enquanto;
      retorna Solução
fim constroiSolucao
```

Figura 3: Pseudocódigo da fase de construção da metaheurística GRASP.
Fonte: SILVA, OCHI, MARTINS, 2006

O processo da busca local por uma solução ótima é a etapa do algoritmo com maior custo computacional. Uma vez encontrado uma solução melhor, a solução inicial

será substituída e uma nova iteração iniciada. Este processo se repetirá até que uma solução melhor não seja encontrada, ou que o número de iterações pré-determinada seja alcançado. Este último critério de parada é normalmente usado a fim de se evitar que o algoritmo entre em um *loop* infinito. A Figura 4 mostra um pseudocódigo para o algoritmo de busca local para a metaheurística GRASP.

```
Proc buscaLocal ()
      enquanto solução não for ótimo local faca
            encontre t na vizinhança de s
            se custo(t) < custo(s) então
                  Solução = t
            fim se
      Fim enquanto;
      retorna Solução
fim buscaLocal
```

Figura 4: Pseudocódigo da fase de busca local da metaheurística GRASP.
Fonte: SILVA, OCHI, MARTINS, 2006

3 UMA NOVA ABORDAGEM DE APRENDIZADO ATIVO UTILIZANDO O PROBLEMA DA DIVERSIDADE MÁXIMA PARA DESCOBRIR FALHAS EM CÓDIGOS FONTE

Nesta seção vamos apresentar o problema da descoberta de falhas no código-fonte e como contribuiremos para solução desse problema.

Como dito anteriormente, a Engenharia de Software vem se destacando no cenário da computação pelo forte crescimento e pela diversidade de técnicas que estão envolvidas em todo seu processo. O grande número de técnicas e métodos envolvidos tem chamado atenção de cientistas de várias áreas da ciência da computação, tornando a Engenharia de Software um campo interdisciplinar repleto de possibilidades para pesquisa. Dentre as áreas que tem se juntado a ES destaca-se a Otimização em Engenharia de Software como uma das mais recentes e, também, em crescente avanço (FREITAS et al, 2009).

A otimização chega para tentar resolver os problemas que surgiram com o crescimento da Engenharia de Software e conseguinte crescimento de seus artefatos. Além dos inúmeros documentos criados durante o processo de desenvolvimento, cresce também o tamanho do código-fonte dos sistemas, uma vez que a tecnologia de *hardware* tem permitido a inserção cada vez maior de recursos. O campo de testes de software tem recebido atenção, mas ainda demanda certas necessidades. Usualmente os testes são executados após uma determinada função ser desenvolvida. Em um sistema de grande porte, onde vários desenvolvedores trabalham em módulos diversos, o responsável pela gerencia dos testes possui pouca, ou nenhuma, informação capaz de direcionar seu trabalho.

Propomos, então, uma solução para esta falta de informação, visando identificar no código-fonte do sistema em desenvolvimento aqueles com possibilidade de falhar. É fácil notar que, analisar todo o código a procura daquele que pode apresentar uma falha e, após, rotular esse código para inspeção, é uma tarefa de custo alto. Para solucionar esse problema, faremos uso de algoritmos de aprendizado de máquina.

A idéia é que o algoritmo realize a tarefa de selecionar o código que possua maior diversidade entre todos os disponíveis para que o mesmo seja classificado, trazendo ganho de informação para o algoritmo de aprendizado. O exemplo com maior diversidade é aquele que o algoritmo tem grandes dúvidas se ele apresentará falhas ou não, porque seu código possui tanto padrões de códigos que falham quanto de códigos

que não falham. Esse exemplo será então rotulado por um usuário ou um oráculo. Conhecendo o rótulo de exemplos duvidosos, o algoritmo de aprendizado acaba ganhando bastante informação. Para o algoritmo, nada adianta que o usuário rotule um exemplo que é obviamente um código com falha ou obviamente um código sem falha. São os que trazem dúvida os mais interessantes a serem rotulados.

Para selecionar código com informação relevante, o algoritmo com Aprendizado Ativo necessita de uma função objetivo. Uma função objetivo fornece ao algoritmo uma métrica para a seleção. Nesse trabalho optamos por uma solução não trivial e inédita, utilizando uma solução para o Problema da Diversidade Máxima como nossa métrica. Escolhidas determinadas características dos códigos rotulados, o algoritmo irá buscar aqueles que mais divergem. O fluxo de aprendizado pode ser visualizado na Figura 5.

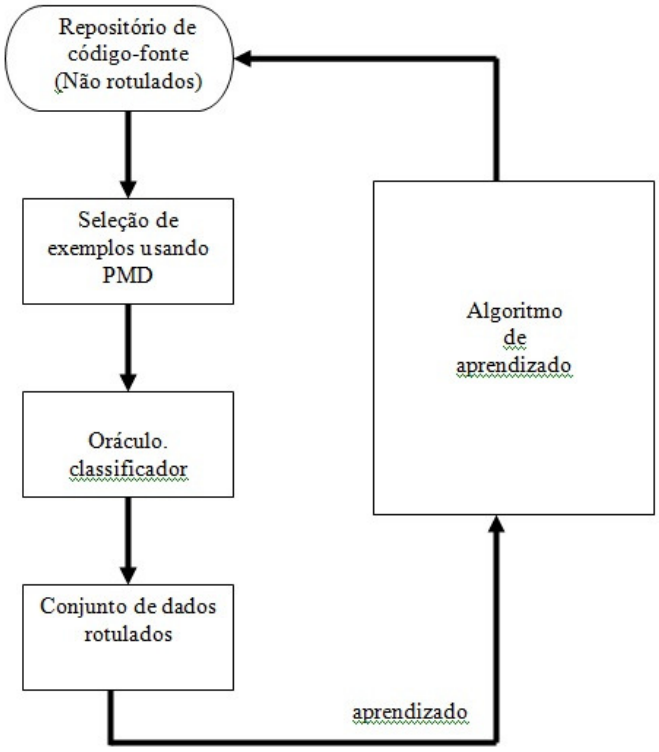

Figura 5: Ciclo de aprendizado
Fonte: Próprio autor

A implementação desse algoritmo está sendo desenvolvida usando a metaheurística *GRASP*. Esta metaheurística já foi usada para a solução do problema da diversidade máxima e mostrou-se computacionalmente viável (SILVA, OCHI, MARTINS, 2006; ANDRADE, 2003). Nosso trabalho prossegue em fases de

experimentos, já que seu principal objetivo era propor a solução, que agora está sendo validada.

4 APLICAÇÕES EM ENGENHARIA DE SOFTWARE

A Engenharia de Software (ES) é a área da computação que estuda as teorias, os métodos, processos e as ferramentas usadas para desenvolver e manter programas. O crescente aumento na complexidade desses métodos e processos, principalmente, trazem consigo o aumento de informações geradas acerca do software e, por muitas vezes, o aumento de linhas de código dado a complexidade dos novos sistemas.

O presente trabalho abordou o uso de Aprendizado Ativo aliado ao Problema da Maximização de Diversidade e como ele pode ser aplicado na ES. Entretanto, as ferramentas usadas para a solução do problema PMD estão ganhando espaço na comunidade científica interessada em ES, um exemplo é o uso de aprendizado de máquina a fim de prever quais módulos provavelmente irão conter a maioria dos defeitos, o que permite aos gerentes de testes otimizarem a alocação de recursos (MENZIES et al, 2010-a); Em um estudo sobre estimativa de esforços para o desenvolvimento de software MENZIES et al (2010-b) usa uma técnica de mineração de dados para isolar os melhores métodos de um conjunto de 158 para serem aplicados em uma determinada aplicação.

Como dito anteriormente, uma nova e promissora área, fruto da união da Engenharia de Software e da Otimização Combinatória, surgiu e tem ganhado espaço, é a Engenharia de Software Baseada em Buscas (*Search-based software engineering*) (FREITAS et al, 2009). Esta nova área surgiu com a demanda de problemas decorrentes do processo de desenvolvimento que exigem uma abordagem automatizada. Problemas como análise de requisitos, otimização de código, estimativa de software, geração e seleção de casos de testes podem ser modelados como problemas de otimização e tratados de forma automática. Uma vez modelados, os problemas podem ser atacados usando um ou mais métodos de otimização, destacando-se os algoritmos denominados de metaheurísticas, tais como algoritmos genéticos, *Simulated Annealing* e *GRASP* (FREITAS et al, 2009).

A área de testes de software é uma das mais beneficiadas com a otimização. Problemas como seleção de casos de teste, geração de dados para teste e a priorização dos casos de testes têm sido amplamente abordados por técnicas da otimização combinatória. Outra área de grande interesse é a estimativa de software, o tamanho do

software, seu custo e a alocação de recursos. Para todos esses problemas já foram propostas soluções usando metaheurísticas que obtiveram resultados computacionalmente viáveis. Uma vez que estes problemas responderam bem aos métodos de otimização, a abordagem proposta nesse trabalho poderá ser aplicada, também, nesses problemas dado que se trata de uma metaheurística cujo aprendizado foi aprimorado.

5 CONCLUSÃO

Este trabalho estudou como métodos de mineração de dados e aprendizado de máquina vem auxiliando engenheiros de software. Em especial, focou em como o aprendizado semi-supervisionado pode ser útil em problemas de aprendizado na área, tais como identificação de falhas em códigos, testes, estimativas de custo, planejamento de projeto, etc.

Durante o estudo do aprendizado semi-supervisionado, mais especificamente da abordagem de Aprendizado Ativo, nos veio a ideia de propor a utilização de um método de otimização para solução do problema de seleção de exemplos a serem rotulados.

Uma breve consulta na literatura mostrou que o método ainda não havia sido proposto. O resultado deste estudo permitiu propor o uso do Problema da Maximização da Diversidade, resolvido por um método GRASP, para solução do problema de exemplos de Aprendizado Ativo.

Esse novo método parece promissor, foi implementado e encontra-se em fase de testes. Sabemos que esse método será útil para resolver diversos problemas da engenharia de software através de aprendizado semi-supervisionado.

Em trabalhos futuros consideramos interessante o emprego desse método em outras áreas da Engenharia de Software como a estimativa de custos e tamanho de software. Outra opção seria a implementação do algoritmo usando outra heurística.

REFERÊNCIAS

ANDRADE, P.M.F.; PLASTINO, A.; MARTINS, S.; OCHI, L.S. **GRASP for the maximum diversity problem.** Proc. of the V Metaheuristic International Conference (V MIC), Kyoto, Japão, 2003.

BASU, S.; BANERJEE, A.; MOONEY, R**. Semi-supervised clustering by seeding**. In Proceedings of the Nineteenth International Conference on Machine Learning - ICML-2002, Sidney, Australia, pp. 19-26, 2002.

FEO, T.; RESENDE, M. **Greedy randomized adaptive search procedures**. Journal of Global Optimization, v. 6, p. 109–133, 1995

FREITAS, F. G.; MAIA, C. L. B.; COUTINHO, D. P.; CAMPOS, G. A. L.; SOUZA, J. T. **Aplicação de Metaheurísticas em Problemas da Engenharia de Software: Revisão de Literatura**. II Congresso Tecnológico Infobrasil (Infobrasil 2009), Fortaleza, 2009.

GHOSH, J. B. **Computational aspects of the maximum diversity problem**, Operations Research Letters v.19 , PP.175–181, 1996.

HAN, J.; KAMBER, M. **Data mining: concepts and techniques**, Morgan Kaufmann Publishers Inc., San Francisco, CA, 2000

MATEUS, G. R.; RESENDE, M. G. C.; SILVA, R. M. A. **GRASP: Procedimentos de busca gulosos, aleatórios, e adaptativos, Manual de computação evolutiva e metaheurística**, Editora UFMG, 2010.

NILSSON, N. J. **Introduction To Machine Learning**, 1998. Disponível em http://robotics.stanford.edu/~nilsson/mlbook.html. Acessado em 03 de Maio de 2011

MENZIES, T.; MILTON, Z.; TURHAN, B.; CUKIC, B.; JIANG, Y.; BENER, A. **Defect prediction from static code features: current results, limitations, new approaches, Automated Software Engineering**, Automated Software Engineering, v.17 n.4, pp.375-407, 2010a.

MENZIES, T.; JALALI, O.; HIHN, J.; BAKER, D.; LUM, K. **Stable Rankings for Different Effort Models**, Automated Software Engineering, v. 17, n.4, pp. 409-437, 2010b.

PRASAD, A.V.; KRISHNA, S. R. **Data Mining for Secure Software Engineering – Source Code Management Tool Case Study** , International Journal of Engineering Science and Technology, v.2, 2010

SETTLES, B. **Active learning literature survey**. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

SILVA, G. C.; OCHI, L. S.; MARTINS, S. L. **Proposta e Avaliação de Heurísticas GRASP para o Problema da Diversidade Máxima**: PESQUISA OPERACIONAL

(Indexada em: Int. Abstracts in Operations Research; e no Statistical Theory and Methods Abstracts), v.26(2), pp. 321-360, 2006.

SILVA, G. C.; ANDRADE, M. R.; OCHI, L. S.; MARTINS, S. L.; PLASTINO, A. "**New Heuristics for the Maximum Diversity Problem**". Journal of Heuristics SPRINGER , v.13, pp: 315-336, 2007.

WAGSTAFF, K.; CARDIE, C.; ROGERS, S.; SCHROEDL, S. **Constrained k-means clustering with background knowledge**. In Proceedings of the Eighteenth International Conference on Machine Learning - ICML-2002, Williamstown, MA, USA, pp. 577-584, 2001.