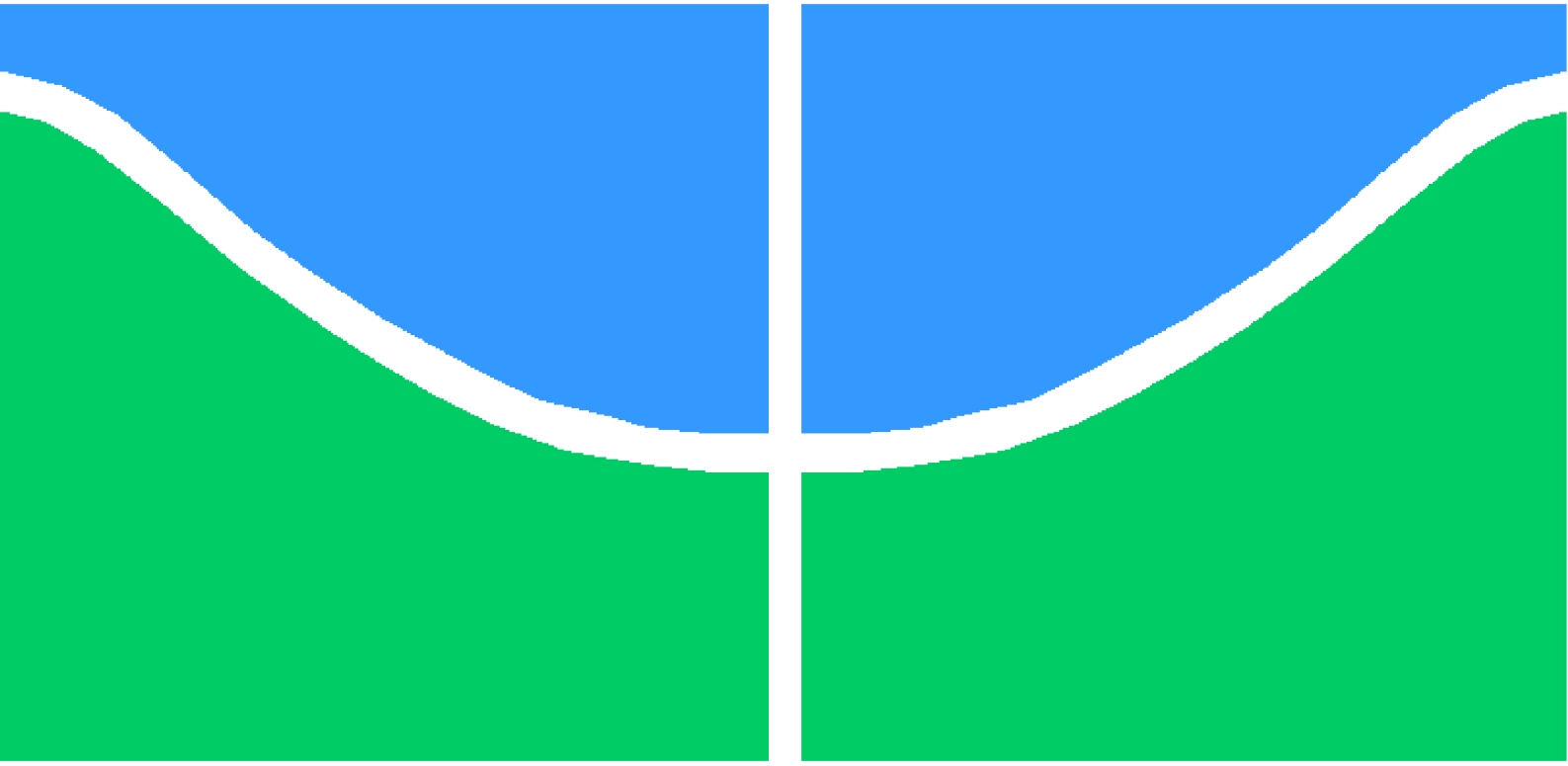

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

Estratégia de contexto temporal em recomendação de pacotes GNU/Linux

Autor: Lucas Albuquerque Medeiros de Moura, Luciano Prestes Cavalcanti

Orientador: Prof. Dr. Paulo Roberto Miranda Meirelles

Brasília, DF

2016

Lucas Albuquerque Medeiros de Moura, Luciano Prestes Cavalcanti

Estratégia de contexto temporal em recomendação de pacotes GNU/Linux

Monografia submetida ao curso de graduação em Engenharia de Softwareda Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Paulo Roberto Miranda Meirelles

Coorientador: Prof. Dr. Edson Alves da Costa Júnior

Brasília, DF

2016

Lucas Albuquerque Medeiros de Moura, Luciano Prestes Cavalcanti
Estratégia de contexto temporal em recomendação de pacotes GNU/Linux/ Lucas Albuquerque Medeiros de Moura, Luciano Prestes Cavalcanti. – Brasília, DF, 2016-
71 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Paulo Roberto Miranda Meirelles

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2016.

1. Sistemas de recomendação. 2. Debian. I. Prof. Dr. Paulo Roberto Miranda Meirelles. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Estratégia de contexto temporal em recomendação de pacotes GNU/Linux

CDU 02:141:005.6

Lucas Albuquerque Medeiros de Moura, Luciano Prestes Cavalcanti

Estratégia de contexto temporal em recomendação de pacotes GNU/Linux

Monografia submetida ao curso de graduação em Engenharia de Softwareda Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Trabalho aprovado. Brasília, DF, 20 de julho de 2016:

Prof. Dr. Paulo Roberto Miranda Meirelles
Orientador

Prof. Dr. Edson Alves da Costa Júnior
Coorientador

Prof. Dr. Fábio Macedo Mendes
Convidado 1

Dr. Antônio Terceiro
Convidado 2

Brasília, DF
2016

Resumo

Sistemas de recomendação estão ficando cada vez mais populares. Diversas aplicações que contêm uma grande gama de itens ou serviços para prover aos seus usuários já estão usando sistemas de recomendação para ajudar usuários na tarefa de melhor acharem suas preferências no sistema, como o *Netflix* e a *Amazon*. Um exemplo onde tal sistema pode ser empregado é em distribuições GNU/Linux, como o Debian, onde o número de pacotes cresce diariamente, contendo atualmente mais de 49.000 pacotes disponíveis para seus usuários. Dessa forma, recomendar o pacote certo para o usuário pode ajudar não só o mesmo na realização de suas atividades, como contribuir para a comunidade como um todo. Esta pesquisa visa então propor o uso de contexto temporal do uso dos pacotes do usuário como diferencial na criação do seu perfil de recomendação, com a hipótese de que tal informação irá prover melhores recomendações. Para isso, essa pesquisa irá usar como base o software *AppRecommender*, que já provê recomendação de pacotes para usuários GNU/Linux.

Palavras-chaves: Sistemas de recomendação, Debian, Aprendizado de máquina.

Abstract

Recommender systems are becoming more popular everyday. Different applications which have a huge number of items or services to provide to their user are now using recommender systems to help users to find their preferences easier on the application, such as Netflix and Amazon. An example of applications that could benefit from recommender systems are GNU/Linux distributions, such as Debian. The number of packages for Debian systems is increasing everyday, with more than 49000 packages available to users. In that context, recommend the right package to the right user may not only help the user on performing his daily task, but also improve the community as a whole. This research aims to use a temporal context of the package use as a differential when creating the user recommendation profile, with the hypothesis that it will provide better recommendations. This will be done based on the software *AppRecommender*, which already provides some package recommendation techniques to Debian users.

Key-words: Recommender systems, Debian, Machine learning

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ROC	*Receiver Operator Characteristic Curve*
TFIDF	*Term Frequency - Inverse Term Frequency*

Sumário

1 Introdução

Segundo Burke (2002), sistemas de recomendação são aqueles que produzem recomendações individualizadas como saída ou que guiam o usuário de forma personalizada para itens interessantes ou úteis em um conjunto grande de opções. Com essa definição em mente, muitas empresas buscam impulsionar tanto a venda quanto a visualização dos itens que possuem, e até mesmo melhor direcionar produtos para o usuário. Exemplos clássicos desse uso são a *Amazon* provendo um loja online praticamente toda personalizada (JANNACH, 2008), e a empresa *Netflix* que usa sistemas de recomendação para prover filmes a seus usuários. Vale ressaltar que esta empresa recentemente realizou um concurso no valor de um milhão de doláres para quem pudesse melhorar a performance do seu sistema de recomendação (KOREN; BELL; VOLINSKY, 2009).

Com essa informação em mente, percebe-se que sistemas de recomendação podem trazer benefícios para grande gama de aplicações em contextos diferentes. Um desses contextos pode ser a recomendação de pacotes GNU/Linux. Um dos sistemas GNU/Linux mais popular é o Debian, que no período de realização deste trabalho possui 49.096 pacotes em seus repositórios [1]. Dado esse número elevado de pacotes, usuários podem ter dificuldades em encontrar pacotes que lhes agradem ou até mesmo ficarem sabendo de algum novo pacote no sistema. Além disso, sistemas de recomendação também podem ajudar a comunidade, pois com mais usuários do pacote, pode-se entender que o número de *bugs* registrados tende a aumentar, podendo assim aumentar a qualidade do software sendo usado.

Baseado neste contexto, um sistema de recomendação de pacotes, chamado *AppRecommender* foi desenvolvido para sistemas Debian pela *Debian Developer* Tássia Camões Araújo durante o seu mestrado na Universidade de São Paulo no ano de 2011. Este sistema visa a recomendação de pacotes baseado em quesitos como os pacotes já instalados pelo usuário ou até mesmo submissões do *popularity-contest*. Dessa forma, o projeto abrange desde recomendações por conteúdo a até mesmo recomendações colaborativas.

Entretanto, dado a gama de pacotes disponíveis em sistemas Debian, prover uma recomendação útil para o usuário pode se tornar um problema. O problema de utilidade das recomendações também se prova presente no *AppRecommender*, onde o resultado das recomendações por conteúdo não se mostrou tão satisfatória (ARAÚJO, 2011). Dessa forma, partindo do pressuposto de que o usuário usa apenas uma parte dos seus pacotes em um dado momento, entender o contexto temporal do uso de pacotes pode acarretar em recomendações reais e mais úteis ao usuário.

[1] Dados coletados 08/11/2015

Afim de colaborar com o sistema Debian, esta pesquisa visa juntar informações temporais de uso de pacotes a um sistema de recomendação de pacotes já existente. Para isso, duas abordagens serão propostas, uma determinística e uma por aprendizado de máquina. Com essas abordagens, a pesquisa visa então entender se o contexto temporal irá trazer melhorias para as recomendações feitas pelo sistema.

1.1 Justificativa

Sabendo do crescente número de pacotes presentes no repositório do Debian e a necessidade de se criar uma comunidade em torno desses pacotes, este trabalho visa ajudar tantos os usuários Debian que gostariam de usar mais pacotes que lhes forneçam funcionalidades interessantes no seu dia-a-dia, quanto a própria comunidade Debian, ao possibilitar que mais usuários usem determinados pacotes, podendo assim aumentar a comunidade em torno do mesmo.

Além disso, esta pesquisa também visa servir de insumo para outros estudos relacionados a recomendação de pacotes GNU/Linux ou até mesmo pesquisas interessadas em aplicar contexto temporal em seus sistemas de recomendação.

1.2 Objetivo

Este trabalho tem como principal objetivo estudar como o contexto temporal pode afetar a recomendação de pacotes GNU/Linux. Nesta etapa inicial do projeto, pretende-se especificar os modelos que serão usados para extrair informação temporal de um pacote, além de modelos de uso dessas informações em um sistema de recomendação.

Para a segunda parte desta pesquisa, o objetivo será a comparação direta dos modelos descritos via um consulta de opinião de usuário quanto a efetividade das recomendações produzidas.

Dessa forma, pode-se dizer que está pesquisa visa responder a seguinte questão-problema:

"É possível melhorar a recomendação de pacotes usando contexto temporal dos pacotes de um usuário"

1.3 Organização do trabalho

Este trabalho está dividido em 3 capítulos distintos. No Capítulo 2, são apresentadas as revisões bibliográficas feitas para sistemas de recomendação e aprendizado de máquina. No Capítulo 3, é apresentada a metodologia proposta para esta pesquisa, contendo desde as hipóteses levantadas para a pesquisa, como o sistema de recomendação

usado como base e como os resultados obtidos serão comparados. Por fim, o último capítulo é usado para relatar algumas considerações do trabalho sendo desenvolvido, como os resultados obtidos até o momento, além de atividades que ainda precisam ser feitas, usando um contexto de um cronograma para relatar tais atividades.

2 Fundamentação Teórica

2.1 Sistemas de recomendação

Sistemas de recomendação são ferramentas computacionais e técnicas usadas para produzir recomendação de itens úteis a um usuário (MAHMOOD; RICCI, 2009). Sendo assim, um sistema de recomendação pode ser usado em diversos contextos, desde melhorar a experiência de usuário a até mesmo melhorar a taxa de vendas em uma aplicação comercial. Além disso, um sistema de recomendação também pode ser usado para recomendar uma grande gama de itens, como filmes e livros (RICCI; ROKACH; SHAPIRA, 2011). Para realizar a tarefa de recomendação, duas abordagens distintas podem ser usadas: recomendação não-personalizadas e personalizadas.

Recomendações não-personalizadas são aquelas que não usam nenhuma informação do usuário para realizar a recomendação. Exemplo de estratégias assim podem ser visto em sites de música que apresentam a lista das 10 músicas mais ouvidas. Já recomendações personalizadas, são aquelas que usam as avaliações já realizadas por um usuário sobre um item para criar uma função preditiva que irá avaliar itens que ainda não foram avaliados pelo usuário. Então, o sistema recomenda aqueles que possuem maior valor apresentado pela função preditiva. A Figura 1 mostra o processo para realizar uma recomendação personalizada (PICAULT et al., 2011).

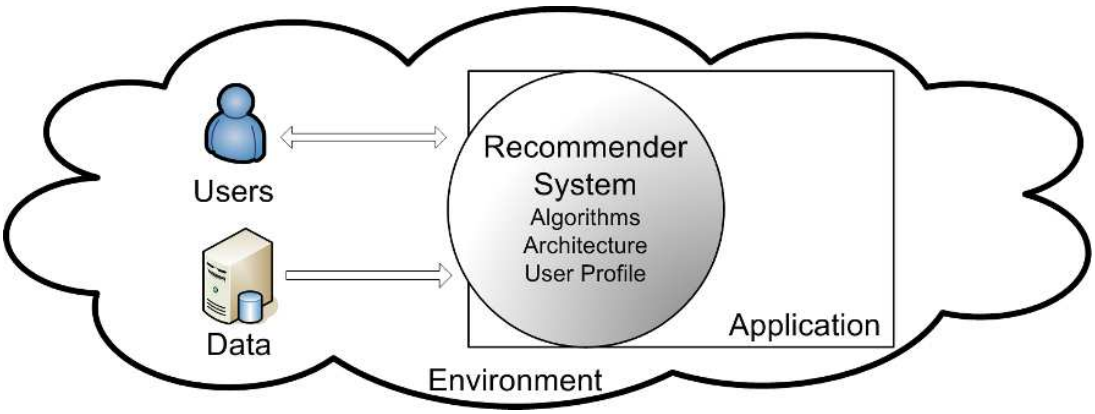

Figura 1 – Modelo para criação de um sistema de recomendação personalizado

Pode-se ver na Figura 1 que as principais entradas para um sistema de recomendação são os usuários do mesmo e os dados relacionados a este mesmo usuário. No que tange o sistema de recomendação em si, pode-se observar fatores importantes no design do mesmo, como o perfil do usuário, o algoritmo de recomendação ou estratégia a ser usada e a infra-estrura a ser usada. Além disso, nenhum sistema de recomendação está imune ao contexto no qual o mesmo será usado.

Vale considerar que o perfil do usuário e a estratégia de recomendação usadas são bastante dependentes um do outro. O perfil do usuário é a forma como o usuário será representado no sistema, podendo ser gerado de inúmeras maneiras, como uma lista de itens avaliados por exemplo. Sendo assim, características dos itens, números de itens avaliados e a forma como o perfil do usuário é modelado são aspectos fundamentais para a seleção de uma estratégia de recomendação, pois dependendo da configuração desses atributos, algumas estratégias de recomendação devem ser descartadas (PICAULT et al., 2011).

2.1.1 Estratégias de recomendação

Com o conjunto de dados definidos, pode-se então escolher uma estratégia de recomendação adequada. Para agrupar tais estratégias, considerou-se taxinomia de alguns autores, como por exemplo (BURKE, 2007). As recomendações descritas a seguir foram as que se mostraram mais presentes nas taxinomias estudadas.

- **Estratégias baseadas em conteúdo:** São aquelas que buscam recomendar itens similares ao que usuário já avaliou. A similiraridade dos itens é estimada pelos atributos que compõem um dado item no sistema. O fluxo de uma estratégia baseada em conteúdo pode ser vista na Figura 2 (LOPS; GEMMIS; SEMERARO, 2011).

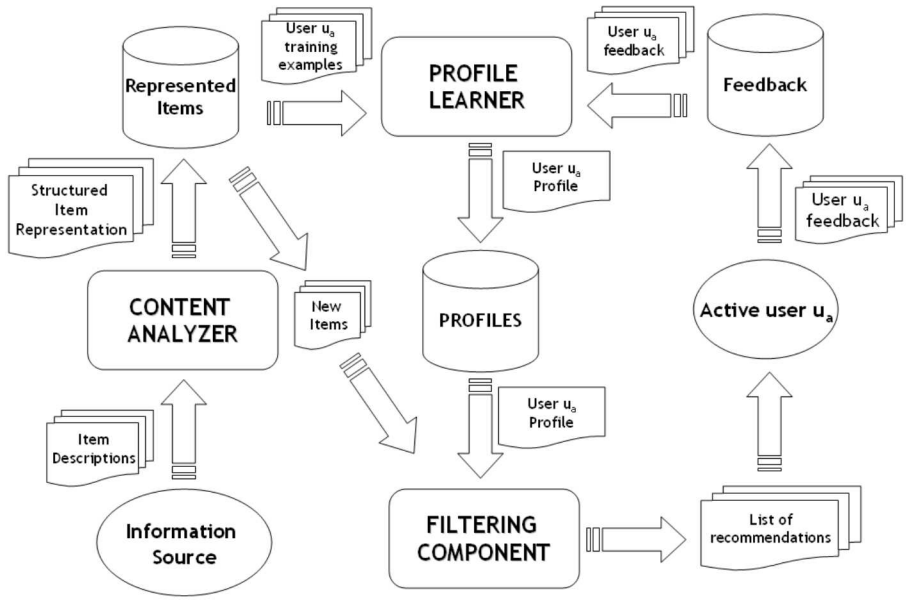

Figura 2 – Fluxo para construção de um sistema de recomendação por conteúdo

Pode-se ver pela Figura 2 que uma recomendação por conteúdo se baseia fortemente em alguns itens chave. O *Content Analyser* é a parte do sistema de recomendação que transforma os itens avaliados pelo usuário em entradas válidas para o algoritmo, ou seja, seleciona os atributos escolhidos do mesmo. O *Profile Learner* é o

responsável por criar o perfil do usuário, analisando os dados filtrados pelo *Content Analyser* e criando um perfil de acordo com um modelo definido pelo sistema de recomendação. Uma técnica normalmente aplicada para essa tarefa é a *Term Frequency-Inverse Term Frequency (TFIDF)* para selecionar os atributos de maior impacto nos itens avaliados pelo usuário, e então compor o perfil do usuário com tais termos (LOPS; GEMMIS; SEMERARO, 2011), essa técnica é explicada no Apêndice A. Por fim, o *Filtering component* nada mais é do que uma ferramenta que recebe o perfil do usuário como entrada e usa o mesmo para filtrar os melhores itens que se aproximam desse perfil da base de dados possível. Sendo assim, a função preditiva desta estratégia de recomendação é uma combinação do perfil de usuário juntamente com uma forma de avaliar a similaridade entre dois itens.

Dado essas características da recomendação baseada por conteúdo, pode-se perceber que a mesma pode sofrer o problema de superespecialização, já que podem haver dificuldades em classificar itens que nunca foram usados pelo usuário e também recomendar apenas itens próximos ao atual perfil do mesmo, dificultando assim a recomendação que podem vir a se tornar uma surpresa para o mesmo. (LOPS; GEMMIS; SEMERARO, 2011).

- **Estratégias colaborativas:** Estratégias que recomendam itens de usuários com avaliações similares aos itens avaliados pelo usuário que está usando o sistema (SCHAFER et al., 2007). Para determinar a similaridade de um usuário a outro, as avaliações de ambos são comparadas, visando assim encontrar usuários próximos um do outro. Uma vez definido a vizinhança do usuário, a recomendação de um item é normalmente feita por selecionar itens mais presentes no perfil dos vizinhos (ARAÚJO, 2011). Vale ressaltar que este tipo de recomendação é uma das mais populares quando se trata de sistemas de recomendação (RICCI; ROKACH; SHAPIRA, 2011).

  Entretanto, vale ressaltar que esse tipo de estratégia necessita fortemente de uma ampla base de usuários para que a mesma possa ser aplicada. Isso se dá pela necessidade de criar uma vizinhança significativa de usuários com gostos similiares. Sendo assim, para aplicações que não possuem uma base de usuário relacionados ou até mesmo tratam do seu usuário de forma isolada, tal estratégia pode ter uma implementação bastante complicada. Além disso, outro problema é que essa estratégia tem um grande problema para recomendar itens que não possuem avaliações por nenhum usuário, o que dependendo da motivação usada para criar o sistema de recomendação, pode vir a se tornar um problema. (RICCI; ROKACH; SHAPIRA, 2011).

- **Estratégias híbridas:** Estratégias que combinam uma ou mais estratégias para gerarem uma recomendação. Normalmente tais estratégias são combinadas para que

os pontos fracos de uma estratégia sejam suavizados pela outra. Por exemplo, combinar estratégias colaborativas com baseadas em conteúdo pode resolver problemas de superespecialização gerados por estratégias baseadas em conteúdo e também resolver problemas de recomendação de itens que ainda não foram avaliados, o que não acontece em recomendações baseadas em conteúdo.

2.1.2 Contexto em sistemas de recomendação

Apesar dos modelos apresentados na Seção 2.1.1 serem bastante distintos um do outro, cada um deles pode usar algumas abordagens diferenciadas para tornar o perfil do usuário mais próximo à realidade. Para isso, uma abordagem que pode ser usada é a de recomendação baseada em contexto. Para esta recomendação, os itens avaliados pelo usuário são combinados com informações de contexto do próprio usuário. Segundo Berry e Linoff (1997), contexto pode ser definido como uma informação que caracteriza os estágios de vida de um usuário e que pode determinar mudanças de preferências e status do mesmo. Informações de contexto podem ser obtidas de forma implícita ou até mesmo serem subentendidas pelo sistema, podendo assim gerar novos atributos interessantes para o perfil do usuário. Um exemplo de informação contextual seria o tempo no qual um usuário realizou a avaliação de um item. Um sistema de recomendação pode usar essa informação para priorizar avaliações mais recentes do que mais antigas, visando manter um perfil mais atualizado do usuário.

Sendo assim, para adicionar informação contextual em um sistema de recomendação personalizado, três abordagens distintas são propostas por Adomavicius e Tuzhilin (2011):

- **Pré-filtragem:** As informações que serão usadas para realizar uma recomendação são pré-filtradas por informações de contexto antes de alimentarem o sistema de recomendação. Um exemplo dessa abordagem pode ser quando um sistema de recomendação só deseja receber as avaliações mais recentes do usuário, usando informação temporal de contexto para filtrar itens avaliados a muito tempo.

- **filtragem pós recomendação:** Após uma recomendação ter sido executada, os itens recomendados são filtrados por informações contextuais. Por exemplo, supondo que um usuário esteja procurando sugestões de restaurantes, o sistema pode fazer essa sugestão e usar a localização do usuário para retirar restaurantes muito longes da sua localização atual.

- **Modelagem contextual:** As informações contextuais são usadas para modelar o perfil do usuário ou até mesmo afetar como o perfil do usuário será criado.

2.1.3 Avaliação de recomendadores

Sistemas de recomendação podem ser avaliados de duas maneiras, que vão desde experimentos de comparação offline, que usam uma base de dados previamente coletada para serem executados, a até mesmo experimentos com usuários, que visam averiguar questões mais subjetivas do recomendador.

Visto a subjetividade intrínseca de um sistema de recomendação, pode-se entender que experimentos de usuários podem prover respostas bastante significativas para algumas questões, entretanto a sua preparação e execução deve ser analisada mais profundamente.

2.1.3.1 Experimentos com usuário

Para experimentos com usuários, um conjunto de usuários é selecionado, visando a interação dos mesmos com o sistema, de forma que eles sejam capazes de responder algumas questões objetivas e até mesmo subjetivas quanto ao processo de recomendação e os próprios itens sendo recomendados.

Entretanto, vale ressaltar que este tipo de experimento apresenta algumas dificuldades. A primeira delas está no custo de se fazer um experimento dessa forma, pois além de fatores como seleção de usuário e isolar um ambiente de teste, tornar tais experimentos repetíveis pode-se tornar bastante complexo (SHANI; GUNAWARDANA, 2011). Além disso, é importante também não informar o objetivo do experimento ao usuário, pois tal informação pode levar ao usuário a se comportar de uma forma diferente, podendo assim gerar resultados comprometidos.

Por fim, este tipo de experimento também permite coletar diversos dados sobre como o usuário utiliza a aplicação, além de poder prover dados qualitativos sobre a própria recomendação, podendo assim responder uma gama bem maior de questões do que os experimentos offline.

2.2 Aprendizado de máquina

Aprendizado pode ser definido como qualquer mudança em um sistema que otimize o seu desempenho na segunda vez que ele repetir a mesma tarefa, ou outra tarefa da mesma população (CUSTóDIO; HENRIQUE, 2010).

O aprendizado de máquina utiliza um princípio de inferência denominado indução, onde através de um conjunto particular de exemplos é possível obter conclusões genéricas (ROBERTO et al., 2011). De um modo abstrato, o aprendizado de máquina funciona como uma caixa preta, onde independente de como é implementado, o mesmo deve ser capaz de encontrar padrões nos dados apresentados e criar um modelo para dados que ainda não foram vistos.

Como exemplo, o aprendizado de máquina pode ser usado para classificar um conjunto de atributos. Exemplos assim são bastante comuns quando se fala em aprendizado de máquina. LeCun et al. (1989) usou algoritmos de aprendizado de máquina para classificar códigos postais escritos a mão. Outro exemplo está no trabalho de Stallkamp et al. (2012), que usa método de aprendizado de máquina para classificar placas de trânsito em diferentes condições.

2.2.1 Aprendizado supervisionado

Quando se fala de aprendizado de máquina, várias técnicas distintas podem ser usadas, entretanto, uma das principais técnicas de aprendizado de máquina é o aprendizado supervisionado, onde é fornecido um treinamento com o conhecimento do ambiente, que é composto por um conjunto de exemplos com entradas e uma saída esperada (ROBERTO et al., 2011).

O objetivo do aprendizado supervisionado é induzir conceitos a partir de exemplos que estão pré-classificados, em outras palavras, exemplos que possuem um rótulo associado a uma classe conhecida (ROBERTO et al., 2011). Utilizado quando se tem tanto as perguntas quanto as respostas, o aprendizado supervisionado é utilizado para se obter uma classificação e funções de aproximação, ou seja, tais algoritmos produzem um modelo à partir de dados já classificados capaz de estimar classificações para dados ainda não vistos.

Utilizando-se do exemplo do algoritmo que classifica um conjunto de atributos, no contexto do aprendizado de máquina supervisionado, para cada entrada disponível no treinamento, está associado um rótulo e valor de um determinado atributo (variando entre 0,1 e 2). O rótulo consiste na classificação de determinado conjunto de atributos. A Tabela 1 mostra um exemplo de algumas entradas de treinamento, com os rótulos e os valores associados a alguns atributos.

Rótulo	**Atributo 1**	**Atributo 2**	**Atributo 3**	**Atributo 4**	**Atributo 5**
1	1	0	1	0	1
2	0	1	1	0	1
0	1	0	0	1	1
1	1	0	1	1	0
2	0	1	1	1	0

Tabela 1 – Entradas de treinamento para o aprendizado de máquina

Após ser realizada a etapa de treinamento, ao receber uma sequência de cinco atributos, o algoritmo deve retornar qual o rótulo, ou seja, a classificação, correspondente a esses atributos. A Tabela 2 mostra um exemplo de entrada para o algoritmo, sendo que

a diferença entre essa entrada para a de treinamento, é que essa não possui o rótulo, pois o rótulo será o resultado da execução do algoritmo.

Atributo 1	**Atributo 2**	**Atributo 3**	**Atributo 4**	**Atributo 5**
1	0	1	1	0

Tabela 2 – Entrada de dados para o algoritmo determinar o rótulo

2.2.2 Classificador Bayesiano

Uma técnica famosa de algoritmos supervisionados é o classificador bayesiano, que é normalmente chamado de bayes ingênuo. Esse algoritmo é baseado no princípio da probabilidade bayesiana. Entretanto, diferente do modelo em si, ele admite que os atributos de um item são sempre independentes um do outro, mesmo que os atributos possuam alguma dependência um do outro (SEGARAN, 2007)

Sendo assim, o classificador bayesiano irá basicamente classificar um item como pertencente a uma classe, se tal classe obter o maior valor de probabilidade dentre os valores de classe possíveis. Dessa forma, pode-se dizer que a classificação bayesiana segue o seguinte formato:

$$Classificador = max(p(C_y) * \prod_{i=1}^{N} p(x_i|C_y))$$

Onde:

- C_y**:** Uma das classes possíveis para classificar um dado;
- **N:** Número total de itens;
- $p(C_y)$**:** Probabilidade da classe "y" dentro do conjunto de dados. Pode ser calculado pela fórmula: $\frac{NC_y}{N_t}$. Onde:
  - NC_y**:** Número de itens da classe "y";
  - N_t**:** Número total de classes;
- $p(x_i|C_y)$**:** Cálculo da probabilidade bayesiana em si, assumindo que as variáveis são independentes. Tal expressão pode ser calculada pela seguinte fórmula: $\frac{Nxi_{C_y}}{NC_y}$. Onde Nxi_{C_y} é o número de vezes que o atributo x_i aparece dentro de um dado marcado como C_y.

Dessa forma, pode-se ver o modelo gerado por esse algoritmo supervisionado é nada mais do que as probabilidades $p(C_y)$ e $p(x_i|C_y)$ para todas as possíveis classificações que serão usadas no problema. Dessa forma, os dados usados para alimentar tal algoritmo são usados para criar exatamente tais valores. Uma vez que os mesmos sejam calculados,

o modelo do algoritmo está completo, podendo o mesmo ser usado para classificar dados ainda não vistos.

Vale ressaltar que o modelo apresentado tem como base a classificação bayesiana usando o modelo de Bernoulli, onde os valores de x_i são valores binários, indicando assim a presença ou não de um atributo em um certo dado. Para valores discretos, pode-se usar outros modelos, como o Gaussiano, onde seria necessário o cálculo da média e variância dos atributos para as dadas classes de classificação (ZHANG, 2004).

Apesar da simplicidade desse algoritmo, alguns cuidados devem ser observados. Um dos principais cuidados é analisar as variáveis, pois quando estas possuem uma grande dependência entre elas, pode levar o modelo a acarretar problemas.

O processo utilizado para implementar o algoritmo do classificador bayesiano é explicado no Apêndice D.

2.2.3 Engenharia de atributos

Considerando que o aprendizado de máquina tem como um alicerce os dados de entrada e principalmente as características desses dados, deve-se ter um cuidado significativo ao selecionar os atributos que serão usados para alimentar um algoritmo de aprendizado de máquina. Essa problemática se torna ainda maior quando um dado apresentado possui um conjunto de atributos muito grande, ou seja, possui uma dimensão alta. Quando isso acontece, pode-se dizer que a densidade entre os dados e as distâncias entre os mesmos se tornam menos significativas (AMATRIAIN et al., 2011). Esse efeito se chama *maldição da dimensionalidade* e pode afetar negativamente um série de algoritmos de aprendizado de máquina.

Um dos efeitos diretos desse problema é o caso chamado de *overfitting*. Esse problema acontece quando um algoritmo fica viciado nas entradas na qual foi treinado e não apresenta resultados satisfatórios quando recebe dados desconhecidos. Uma das formas de evitar tal problema é selecionar bem os atributos de um projeto ou até mesmo usar técnicas para diminuir a dimensionalidade de uma variável de entrada.

Caso a seleção seja manual, é recomendável que a mesma seja realizada ao lado de um especialista na área na qual o aprendizado de máquina será utilizado. Isso se dá pela capacidade de um profissional da área em informar que atributos são relevantes para a classicação de um determindado objeto.

Para a seleção manual de atributos, pode-se ter casos onde a classificação dos dados nunca foi feita ou até mesmo a dificuldade em se encontrar um especialista na área para ajudar na seleção. Mesmo com essas dificuldades, essa etapa é crucial para um bom funcionamento do algoritmo de aprendizado de máquina, principalmente quanto a escalabilidade do mesmo.

2.2.4 Validação cruzada

Existem algumas formas de validar se o algoritmo não está sofrendo *overfitting* e o seu oposto, *underfitting*, onde o algoritmo apresenta elevado valor de *bias* para as entradas usadas como teste, ou seja, o modelo criado pelo algoritmo é muito simples. Uma forma de observar a ocorrência desses dois fenômenos é pela técnica estatística de validação cruzada. Essa técnica é comumente usada para avaliar modelos preditivos, e se baseia na separação dos dados existentes em uma parte de treinamento e uma parte de teste. O conjunto de dados de treinamento é usado para alimentar o algoritmo. Após o treinamento, o conjunto de teste é então usado para validar o algoritmo. (ARAÚJO, 2011).

Uma outra abordagem recorrente quando se fala em validação cruzada é a execução da mesma por um número definido de vezes. Isso acontece com o intuito de aumentar a confiança nos resultados apresentados, pois com maior repetição do algoritmo, pode-se esperar que os conjuntos de treinamento e teste serão sempre diferentes, fazendo com que a média do resultado final seja mais robusta em relação a realidade.

Para verificar um resultado, uma das métricas mais simples é a comparação item a item dos resultados esperados pelos resultados obtidos, que pode ser obtida por:

$$PA = \frac{NVP}{NIT} \tag{2.1}$$

Onde:

- **PA:** Porcentagem de acertos do algoritmo de aprendizagem de máquina.
- **NVP:** Numero de verdadeiros positivos, que se trata do número de itens classificados corretamento pelo algoritmo.
- **NIT:** Numero de itens de teste, ou seja, o números de itens contido no conjunto de dados de testes

Entretanto, essa métrica deve ser usada com cautela, pois caso um conjunto de dados apresente muito mais um rótulo do que outros, tal métrica pode passar orientação errada quanto a real acurácia do algoritmo. Por exemplo, dado um conjunto de teste onde 95% dos dados são rotulados como “X“ e o resto rotulado como “Y“, um classificador que sempre rotula um dado como “X“ obterá uma resultado de 95% de acurácia, apesar de não ser nada útil.

3 Metodologia

Este capítulo é destinado à apresentar toda a metodologia aplicada nesta pesquisa. A metodologia está sendo baseada na definição proposta por Silva e Menezes, que afirmam que toda pesquisa tem como objetivo encontrar respostas para hipóteses propostas (SILVA; MENEZES, 2005)

Com o intuito de classificar a pesquisa sendo desevolvida, observou-se que a mesma pode ser enquadrada como aplicada, quantitativa, exploratória, bibliográfica e experimental (SILVA; MENEZES, 2005).

3.1 Trabalhos relacionados

Durante a pesquisa deste trabalho, encontrou-se alguns trabalhos que também estudam recomendação baseada em contexto, sendo o tempo um dos principais atributos do contexto. Para Lee, Park e Park (2008), o tempo é usado para gerar avaliação implícita para um histórico de compras de diversos usuários. Tal abordagem foi comparada com um modelo de recomendação colaborativa tradicional, usando avaliação explícita dos usuários, sem considerar o tempo. Segundo a pesquisa, percebeu-se um aumento de precisão quando o tempo foi usado como parâmetro do perfil de usuário.

Outra pesquisa relacionada foi a de Ding e Li (2005). Nesta pesquisa, um sistema de recomendação colaborativo sensível ao tempo foi criado. O sistema proposto usava uma função de decaimento exponencial para priorizar itens recentemente avaliados e diminuir o peso de itens avaliados a mais tempo na recomendação. Quando comparado a um sistema de recomendação colaborativa clássico, percebeu-se um aumento de acurácia para o recomendador sensível ao tempo.

Além da comparação de sistema de recomendação, foi também encontrado um trabalho que propõe um modelo diferente do clássico uso do tempo em recomendação, como o usado em Ding e Li (2005). Este modelo foi proposto no artigo de Basile et al. (2015), onde basicamente itens mais antigos não tem seu peso puramente descartado, e sim caso seu conteúdo seja diferente dos novos itens sendo recomendados. Baseado nos testes realizados, percebeu-se que criar o modelo de usuário dessa forma apresenta vantagens sobre a forma padrão de se usar o tempo para recomendação. Entretanto, o trabalho ainda precisa ser testado em bancos de dados de maior escala para poder aferir melhor seus resultados

Por fim, um famoso trabalho que usa o contexto de tempo para recomendação se dá na pesquisa de Koren (2010), que apresenta o modelo de recomendação vencedor do

concurso criado pelo *Netflix* para aumentar a acurácia de suas recomendações de filme. A pesquisa citada usa como diferencial a análise do tempo da avaliação dos filmes realizados pelos usuários em um modelo similar a pesquisa de Basile et al. (2015)

3.2 Planejamento da pesquisa

Tendo como base os conceitos de recomendação apresentados e a forma dinâmica com que usuários instalam e usam novos pacotes em seus próprios sistemas, esta pesquisa visa então responder a seguinte questão problema:

É possível melhorar a recomendação baseada em conteúdo de pacotes para usuários de sistemas GNU/Linux levando em consideração o contexto temporal de uso dos seus pacotes já instalados

Vale ressaltar que para responder esta questão problema, um conjunto de hipóteses foi criada juntamente com um fluxo de trabalho, conforme pode ser visto na Figura 3. As hipóteses e o fluxo definido serão explicados nas seções à seguir.

3.2.1 Hipóteses

Considerando a questão problema definida, as seguintes hipóteses foram definidas como forma de validação:

- ***h1: Ao aumentar o peso de pacotes recentemente usados no sistema, o perfil do usuário gerado será mais preciso, ocasionando assim uma recomendação mais apropriada.*** Ao usar o tempo de uso dos pacotes no sistema, e priorizar os pacotes mais recentemente usados na criação do perfil, acredita-se que a mesma irá produzir um perfil mais consistente do usuário, gerando assim recomendações mais efetivas.
- ***h2: O perfil de usuário gerado por aprendizado de máquina aplicado a uma estratégia de recomendação irá prover melhor resultados que uma abordagem determinística para criação do perfil.*** Considerando os dados presentes em pacotes Debian que podem ser usados para prover a recomendação, pode-se entender que o uso de uma abordagem por aprendizado de máquina irá prover melhores resultados, devido à possível quantidade de atributos que podem caracterizar um pacote.

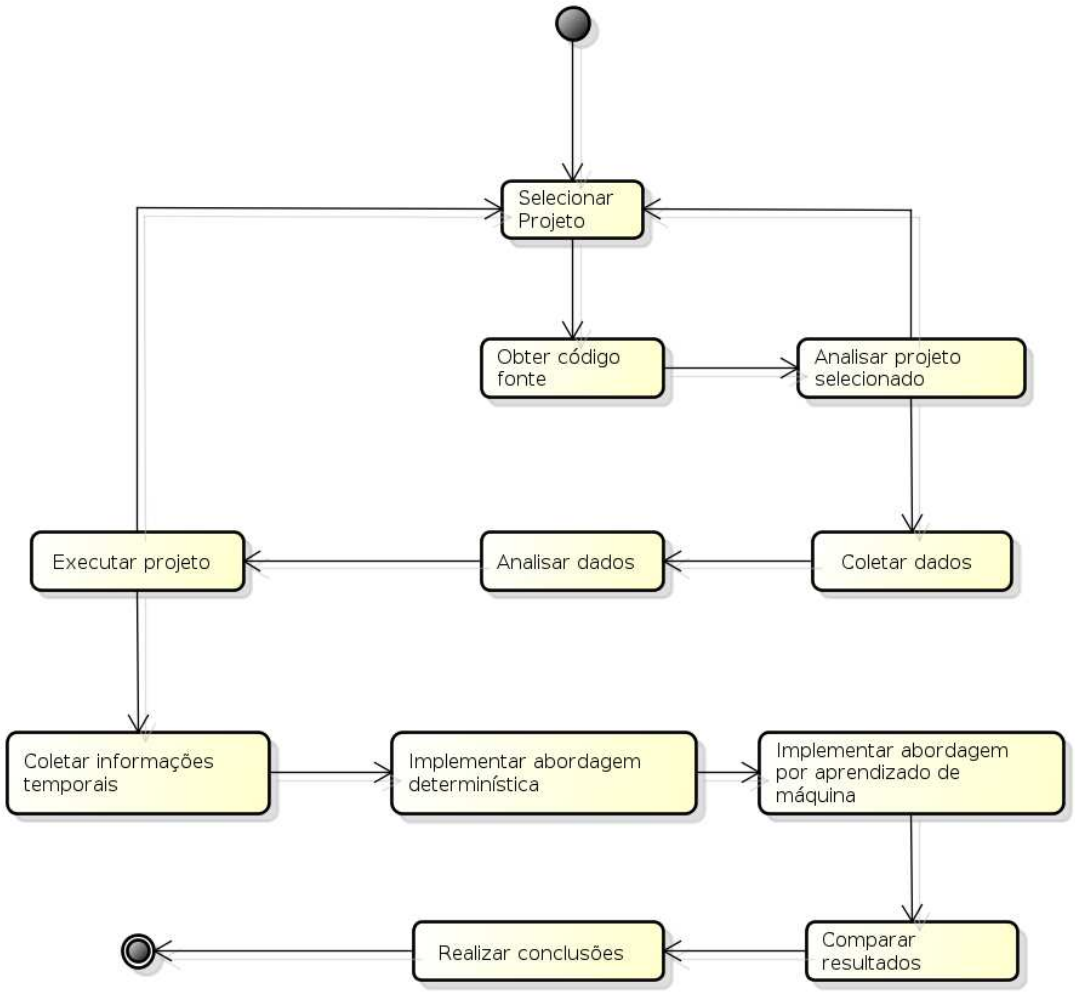

Figura 3 – Processo usado para realização das atividades dessa pesquisa

3.2.2 Seleção do projeto

Para validar as hipóteses levantadas, faz-se necessário escolher um projeto de software livre que implemente um sistema de recomendação de pacotes para sistemas GNU/-Linux e que não use nenhuma informação de contexto em suas recomendações, de forma a permitir a comparação das abordagens propostas com as já implementadas pelo projeto.

3.2.3 Descrição do projeto selecionado

O projeto selecionado que atendia os requisitos propostos para a pesquisa foi o *AppRecommender*, um sistema de recomendação de aplicativos GNU/Linux, cuja principal característica é recomendar aplicações, onde "dada a lista de programas instalados em determinado sistema, o recomendador retorna uma lista de aplicativos sugeridos, que supostamente são aplicativos de potencial interesse para os usuários daquele sistema" (ARAÚJO, 2011). A Figura 4 mostra o funcionamento do *AppRecommender* no terminal de um usuário.

```
lucas@lucas:~/unb/tcc/AppRecommender/bin(master) $ ./apprec.py -s cb
INFO: Set up logger
INFO: Basic config
INFO: Loading recommender filters
INFO: Setting recommender strategy to 'cb'
INFO: Computation started at 2016-06-25 22:15:45.204041
INFO: Recommending applications for user local-2016-06-25 22:15:45.476079
INFO:
 0: ipython-qtconsole
 1: ipython3-qtconsole
 2: ipython-notebook
 3: ipython3-notebook
 4: ipython-notebook-common
 5: osc
 6: mini-buildd-common
 7: python-mini-buildd
 8: cdbs
 9: python3-ipython-genutils
10: python-ipython-genutils
11: fsvs
12: mathematica-fonts
13: science-economics
14: topgit
15: matlab-gdf
16: pybit-web
17: python3-ipdb
18: matlab-support-dev
19: mwrap

INFO: Computation completed at 2016-06-25 22:15:45.625157
INFO: Time elapsed: 0 seconds.
```

Figura 4 – Exemplo de execução da aplicação *AppRecommender*

O *AppRecommender* possui três categorias para estratégias de recomendação, onde cada estratégia é uma forma diferente de se obter uma lista de aplicativos sugeridos através de uma lista de pacotes instalados no sistema. Essas categorias são: baseadas em conteúdo, colaborativas, e híbridas. Segue abaixo as especificações de cada categoria de estratégia:

- ***Baseadas em conteúdo:*** Esse tipo de categoria de estratégia visa montar um perfil de usuário através da lista dos pacotes instalados no sistema, onde o mesmo consiste em uma lista composta por *Debtags* ou termos contidos nas descrições de pacotes que são definidas pelo algoritmo de estratégia escolhida. Através dessa lista, a base de dados com todos os pacotes é consultada e o resultado da recomendação são os pacotes mais relevantes para o perfil criado.

- ***Colaborativas:*** Essa estratégia de recomendação visa a sugestão de novos pacotes via análise de outros usuários com perfis semelhantes ao do usuário, ou seja, tal estratégia visa recomendar novos pacotes de acordo com os pacotes instalados dos vizinhos mais próximos ao perfil do usuário.

- ***Híbridas:*** Essa categoria de estratégia utiliza as duas anteriores, onde há formas diferentes de combiná-las afim de se obter diferentes recomendações.

Para as recomendações colaborativas e híbridas, o *AppRecommender* utiliza uma sistemática para aumentar o peso de pacotes baseados no seu tempo. Entretanto esse

não é o caso para recomendações puramente baseadas em conteúdo, fazendo assim que o mesmo se encaixe nos pré-requisitos levantados para o projeto.

O projeto também utiliza testes de usuário para validar as recomendações implementadas. Tais testes de usuário visam analisar principalmente a precisão e novidade das recomendações.

A licença do *AppRecommender* é GNU GPL e o projeto está disponível em repositório público, permitindo assim que qualquer pessoa possa visualizar o código fonte e contribuir com o projeto. O código fonte da aplicação pode ser encontrado no repositório de projetos Github [1].

As modificações propostas nesta pesquisa foram realizadas em uma cópia local do mesmo, sendo essa mantida pelos pesquisadores deste trabalho no Github [2].

3.3 Coleta de dados

Nesta seção, serão descritos os dados necessários para a execução do *AppRecommender*, juntamente com os dados necessários para a coleta de informação contextual de tempo dos pacotes do usuário durante a recomendação.

3.3.1 AppRecommender

Para o projeto selecionado, observou-se a necessidade de diversos tipos de dados, que podem ser vistos na Tabela 3. Tal tabela mostra os dados usados no *AppRecommender* e como os mesmos podem ser obtidos.

Conforme pode ser observado na Tabela 3, muito dos dados necessários para a execução do *AppRecommender* podem ser obtidos de forma automatizada por scripts de coleta. Entretanto, este não é o caso para as submissões do *popularity-contest*. Isso se dá pelo fato da aplicação não disponibilizar as submissões propriamente ditas, e sim as análises estatísticas feitas em cima das mesmas. Para o *AppRecommender*, tais dados foram obtidos por intermédio de um *Debian Developer*, sendo que algumas questões relacionadas a privacidade dos usuários se tornou necessária. (ARAÚJO, 2011). Tal problemática dificulta bastante o uso das estratégias colaborativas do *AppRecommender*, pois as mesmas dependem desse banco de dados de submissões do *popularity-contest*.

Uma observação também importante se dá na forma como a lista de pacotes válidos é obtida, pois a mesma é obtida conforme as configurações do *sources.list* do usuário. Este arquivo é responsável por dizer ao gerenciador de pacotes onde o mesmo irá encontrar os

[1] <https://github.com/tassia/AppRecommender>
[2] <https://github.com/TCC-AppRecommender/AppRecommender>

Tabela 3 – Dados necessários para execução do AppRecommender

Dados	**Contexto**	**Coleta**
Lista de nome dos pacotes válidos	Esta lista irá ser usada na construção de um índice de busca usado para buscar os pacotes que podem ser recomendados	Coleta automatizada via *script*
Índice de busca para os pacotes do usuário	Usado tanto para estratégias de recomendação por conteúdo, colaborativas e híbridas	Criação automatizada via *script*
Debtags válidas	Usado para filtrar os pacote que serão usados na recomendação	Coleta automatizada via *script*
Submissões do *popularity contest*	Usada para criar um índice de busca contendo as submissões do popularity-contest de todos os usuários que possuem uma submissão	Dados precisam ser manualmente coletados
Indíce de busca para submissões do *popularity contest*	Usado tanto para a realização de recomendações colaborativas e híbridas, assim como para a execução das coletas de opinião usados para avaliar as soluções desenvolvidas.	Criação automatizada via *script*

arquivos binários ou arquivos fonte dos pacote. O código abaixo mostra o conteúdo deste arquivo:

```
deb http://ftp.us.debian.org/debian/ sid main
deb-src http://ftp.us.debian.org/debian/ sid main
```

Conforme visto no código acima, pode-se ver que o mesmo indica pelo rótulo *deb* onde irá puxar os arquivos binários e pelo *deb-src* onde o mesmo irá puxar os arquivos fonte. Além disso, após o link de cada um dos rótulos, existem também atributos, sendo eles a distribuição do usuário e os componentes da mesma. Na figura, pode-se ver que o usuário está usando a distribuição *unstable* do Debian. Além disso, o Debian pode possuir três tipos de pacotes, sendo eles:

- ***Main:*** pacotes totalmente software livre;
- ***Contrib:*** pacotes software livre, mas que dependem de softwares não livres;
- ***Non-free:*** pacotes que não são software livre;

Sendo assim, como a lista de pacotes válidos é obtida localmente, a mesma irá ser baseada tanto na distribuição que o usuário usa, como dos componentes que o mesmo possui no seu *sources.list*.

Por fim, vale ressaltar que foi identificado que as recomendações colaborativas dependiam também de um serviço fornecido pela infra-estrutura do Debian, chamado de *Debian Data Export*, que era usado para facilitar a criação de *queries* no *Ultimate Debian Database*. Entretanto, este serviço se encontra atualmente fora do ar, e, por isso, não é considerado na execução da pesquisa.

3.3.2 Usuários

De forma a melhor avaliar como o contexto temporal está sendo usado pelas novas estratégias, decidiu-se junto com a coleta de opinião dos usuários, coletar as seguintes informações:

- ***Pacotes instalados:*** Necessário para se ter conhecimento dos pacotes que o usuário utiliza;
- ***Pacotes manualmente instalados:*** Estes serão coletados para filtrar as preferências do usuário, para que quando comparado aos pacotes instalados se possa diferenciar pacotes que são automaticamente instalados dos que foram manualmente instalados;
- ***Tempo de acesso e de modificação dos pacotes:*** Estes dados serão usados afim de obter informação quanto à classificação do pacote em relação ao tempo que este foi utilizado;
- ***Caminho do binário de cada pacote:*** Dados coletados afim de se saber qual o binário responsável pela execução do pacote;
- ***Versão do Kernel:*** Necessário para agrupar as recomendações e preferências entre usuários com mesmo kernel;
- ***Distribuição do sistema:*** Necessário para agrupar as recomendações e preferências entre usuários com a mesma distribuição.
- ***Submissão do popularity-contest:*** Informação que é utilizada para permitir a comparação do contexto temporal implementado na pesquisa e o usado pelo *popularity-contest*.

O Apêndice B explica como os dados são coletados.

3.4 Análise dos Dados

Este seção irá mostrar a análise dos dados que são usados para prover uma recomendação propriamente dita. Além disso, também serão apresentados os novos dados que esta pesquisa se propõe a anexar as atuais recomendações.

3.4.1 Recomendação baseada em conteúdo

Para recomendações baseadas em conteúdo, é necessário analisar tanto os dados que são usados para construir o perfil do usuário quanto para realizar a recomendação propriamente dita.

Para a construção do perfil do usuário, os pacotes instalados pelo usuário são usados. Sendo assim, nenhum pacote possui uma avaliação explícita no sistema, ou seja, a avaliação do item de recomendação nesse caso consiste apenas no fato do usuário possuir o pacote instalado em seu sistema.

Para analisar um pacote Debian, pode-se observar o arquivo *control* que está presente em todo pacote Debian. Este arquivo armazena todas as informações que são usadas para a construção do perfil do usuário. A Figura 5 mostra um exemplo desse arquivo.

```
Package: vim
Version: 2:7.4.826-1
Installed-Size: 2206
Maintainer: Debian Vim Maintainers <pkg-vim-maintainers@lists.alioth.debian.org>
Architecture: amd64
Provides: editor
Depends: vim-common (= 2:7.4.826-1), vim-runtime (= 2:7.4.826-1), libacl1 (>= 2.2.51-8), libc6 (>= 2.15), libgpm2 (>= 1
.20.4), libselinux1 (>= 1.32), libtinfo5
Suggests: ctags, vim-doc, vim-scripts
Description-en: Vi IMproved - enhanced vi editor
 Vim is an almost compatible version of the UNIX editor Vi.
 .
 Many new features have been added: multi level undo, syntax
 highlighting, command line history, on-line help, filename
 completion, block operations, folding, Unicode support, etc.
 .
 This package contains a version of vim compiled with a rather
 standard set of features.  This package does not provide a GUI
 version of Vim.  See the other vim-* packages if you need more
 (or less).
Description-md5: 59e8b8f7757db8b53566d5d119872de8
Homepage: http://www.vim.org/
Tag: devel::editor, implemented-in::c, interface::commandline,
 interface::text-mode, role::program, scope::application,
 uitoolkit::ncurses, use::editing, works-with::text, works-with::unicode
Section: editors
Priority: optional
Filename: pool/main/v/vim/vim_7.4.826-1_amd64.deb
Size: 962554
MD5sum: 58a99024595c292861ef39484b659ea4
SHA1: 74d47f71478b323654309db6e4657d237508b0dd
SHA256: 31e72832109f030183dfb9a41731a841d491b77def184619bab463716572e64c
```

Figura 5 – Exemplo de um arquivo control de um pacote Debian

Os principais dados usados para compor o perfil do usuário são a descrição e as *Debtags* de cada pacote instalado.

As *Debtags* são basicamente uma forma de melhor categorizar um pacote fora das 33 seções disponíveis no Debian, além também de permitir que um pacote seja categorizado em diferentes frentes ao mesmo tempo (ZINI, 2005). Isso pode ser visto na Figura 5, onde o software *vim* possui as *Debtags devel::editor* e *interface::commandline*, ou seja,

está se enquadrando tanto como um editor de texto e uma aplicação que roda em linha de comando.

O processo de criação de *Debtags* é manual, ou seja, um usuário deve associar uma *Debtag* a um pacote e a comunidade Debian deve verificar se aquela *Debtag* sugerida condiz com a aplicação em si. Atualmente, existem 45432 pacotes categorizados com Debtags [3].

Por fim, vale lembrar que nem todas as *Debtags* são consideradas significativas para o processo de recomendação, sendo que uma filtragem é realizada visando selecionar apenas as *Debtags* válidas para a construção do perfil do usuário (ARAÚJO, 2011).

A segunda fonte de dados usada é a descrição do pacote, sendo no caso cada termo individual. A descrição de um pacote é realizada pelo empacotador na hora com que o mesmo cria o arquivo *control*.

Com essas duas informações, o *AppRecommender* cria então um perfil de usuário onde os termos de maior peso são usados para compor tal perfil. Isso é feito por algoritmos como *Term Frequency Inverse Document Frequency Sub-Linear* (TFIDF-sublinear).

O perfil de usuário é então uma lista contendo os termos e *Debtags* de maior peso. Este perfil é usado então para consultar toda a base de pacotes do usuário, baseado em seu *sources.list*. Isso é realizado através do *apt-xapian-index*, uma aplicação que provê uma forma de buscar pacotes no Debian por meio de *queries*, sendo que no caso do *AppRecommender*, a *query* de busca é o perfil gerado para o usuário. Dessa forma, a *query* gerada é composta por cada item do perfil do usuário, sendo que a aplicação retorna os resultados que apresentam o maior número de termos passados na *query*. Vale ressaltar que este pacote é usado para construir todos os índices de busca apontados pela Tabela 3. Finalmente, os pacotes retornados são a recomendação final da aplicação.

Com isso dito, o *AppRecommender* implementa estratégias de recomendação que misturam algum dos dados na geração do perfil do usuário. Existem estratégias que usam apenas *Debtags*, enquanto outras combinam de forma balanceada termos da descrição com *Debtags* no perfil do usuário gerado.

3.4.2 Recomendação colaborativa

Os principais dados usados na recomendação colaborativa são as submissões do *popularity-contest*. O *popularity-contest* é um sistema usado pelo Debian para acompanhar o uso dos pacotes sendo mantidos pelo sistema e ajudar também na escolha dos pacotes que devem ser selecionados para o CD de instalação do Debian, onde os mais populares são escolhidos (ARAÚJO, 2011).

No momento que o usuário está instalando o sistema Debian, o mesmo é questi-

[3] Dados coletados no dia 06/12/2015

onado se gostaria de participar do *popularity-contest*. Caso o usuário diga sim, o mesmo irá enviar periodicamente uma submissão ao *popularity-contest* contendo informações de todos os seus pacotes e o tempo de acesso e modificação de acada um deles. A Figura 6 mostra uma submissão típica.

```
POPULARITY-CONTEST-0 TIME:1443922214 ID:ca04b4f431a227f93914b4be22ca014c ARCH:amd64 POPCONVER:1.64 VENDOR:Debian
1443916800 1443787200 libgstreamer-plugins-base1.0-0 /usr/lib/x86_64-linux-gnu/libgstapp-1.0.so.0.600.0
1443916800 1443787200 libpython2.7-stdlib /usr/lib/python2.7/lib-dynload/_ctypes.x86_64-linux-gnu.so
1443916800 1438862400 xfce4-power-manager /usr/bin/xfce4-power-manager
1443916800 1427371200 libxcb-present0 /usr/lib/x86_64-linux-gnu/libxcb-present.so.0.0.0
1443916800 1443787200 libwebkit2gtk-4.0-37 /usr/lib/x86_64-linux-gnu/libwebkit2gtk-4.0.so.37.11.3
1443916800 1427371200 gir1.2-notify-0.7 /usr/lib/girepository-1.0/Notify-0.7.typelib
1443916800 1438862400 libgcr-ui-3-1 /usr/lib/x86_64-linux-gnu/libgcr-ui-3.so.1.0.0
1443916800 1441843200 acpid /usr/sbin/acpid
1443916800 1428580800 fonts-freefont-otf /usr/share/fonts/opentype/freefont/FreeSerif.otf
```

Figura 6 – Exemplo de um arquivo de submissão para o *popularity-contest*

Conforme pode ser observado na Figura 6, após a primeira linha de cabeçalho, cada linha da submissão possui o formato descrito na Tabela 4.

Campo	**Descrição**
Tempo de acesso	Última vez que o arquivo foi acessado
Tempo de mudança	Última vez que informações do *inode* do arquivo foi modificada
Nome	Nome do pacote
Caminho do pacote	Local onde o pacote se encontra no sistema
Tag	Campo opcional que pode classificar o pacote com algumas considerações temporais, como se o mesmo é recente ou antigo

Tabela 4 – Descrição dos campos de uma submissão ao popularity-contest

Baseado nestas informações, o perfil do usuário em uma recomendação colaborativa é criado baseado em uma submissão *popularity-contest*, extraindo do mesmo os pacotes usados e as informações temporais do mesmo.

Para a recomendação propriamente dita, os vizinhos mais próximos do usuário são aqueles com submissões mais próximas do perfil do usuário. Sendo assim, para as recomendações colaborativas, é necessário conter as submissões de vários usuários do *popularity-contest*, para que a vizinhança criada seja significativa.

3.4.3 Contexto temporal

Dado esse conjunto de dados usados no *AppRecommender*, a pesquisa resolveu adicionar o contexto temporal apenas em recomendações baseadas em conteúdo. Isso se dá pelo fato das recomendações colaborativas já usarem tal atributo e, principalmente, pela dificuldade na obtenção dos dados necessários para execução das estratégias colaborativas.

Com isso dito, o novo atributo que esta pesquisa pretende adicionar é o contexto de uso de um pacote, ou seja, a última vez que o mesmo foi usado. Em distribuições

Debian, isso pode ser verificado via comando *stat*, que recebe como parâmetro o caminho para um pacote. A saída deste programa pode ser visualizada na Figura 7.

```
  File: ‘/usr/bin/vim’ -> ‘/etc/alternatives/vim’
  Size: 21              Blocks: 0          IO Block: 4096   symbolic link
Device: 802h/2050d      Inode: 4737909     Links: 1
Access: (0777/lrwxrwxrwx)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2015-10-12 08:39:26.837183470 -0300
Modify: 2015-03-26 18:40:20.827875691 -0300
Change: 2015-03-26 18:40:21.127875688 -0300
 Birth: -
```

Figura 7 – Exemplo de saída do comando *stat* para o executável do vim

Pode ser visto na Figura 7 que o comando exibe os seguintes atributos de tempo de um dado arquivo (HAAS,):

- ***Access:*** Última vez que um arquivo foi acessado, ou seja, a última vez que seu conteúdo foi de fato acessado.
- ***Modify:*** Última vez que conteúdo de um arquivo foi modificado.
- ***Change:*** Última vez que o status de um arquivo foi modificado, como mudar as permissões do arquivo ou seu dono, ou seja, modificar o *inode* do arquivo.

Dessa forma, pode ser visto que o campo *Access* de um arquivo é um dos principais indicadores de uso de um pacote. Entretanto, o sistema de arquivos usado pode apresentar uma restrição neste campo. Isso acontece devido a como o sistema de arquivos é configurado, pois algumas *flags* especiais podem ser usadas para alterar como esse campo é preenchido [4]:

- ***noatime:*** O sistema de arquivos não irá computar o tempo de acesso do arquivo.
- ***stricatime:*** O sistema de arquivos irá sempre alterar o tempo de acesso do arquivo.
- ***relatime:*** O sistema de arquivos só irá computar o tempo de acesso de um arquivo se o valor desse campo for menor que os valores dos campos *Modify* e *Change*. Esse opção é *default* desde o kernel 2.6.30. Entretanto, novas adições foram feitas, permitindo que o tempo de acesso seja modificado se o mesmo for mais antigo que um dia.

O uso do relatime passou a ser default para montar o sistema de arquivos devido aos problemas de perfomance ocasionados pelo atime. Segundo Ingo Molnar, “atualizar o valor de atime é de longe uma das maiores deficiências de performance que o Linux

[4] https://wiki.debian.org/fstab

possui hoje" (CORBET, 2007). Sendo assim, as informações de contexto que serão usadas, tempo de acesso, modificação e mudança, são aproximações e não refletem perfeitamente o contexto de uso dos pacotes.

Com isso dito, a informação temporal de cada pacote será obtida pela seguinte fórmula:

$$PTP = \frac{TAC - TM}{TAT - TM} \tag{3.1}$$

Onde:

- **PTP:** É o peso temporal do pacote, que se trata do valor obtido pelo resultado da equação.
- **TAC:** É o tempo do ultimo acesso ao pacote, ou seja, quando um pacote foi executado pela última vez, esse tempo é dado em segundos.
- **TM:** É o tempo de modificação, se trata de quando o pacote foi modificado, esse tempo também é dado em segundos.
- **TAT:** É o tempo atual no qual o usuário executa a aplicação, em segundos.

A equação apresentada irá retornar um valor de 0 a 1, sendo que quanto mais perto de 1, mais recente é considerado o arquivo. Vale ressaltar que a fórmula também leva em conta o tempo de modificação, pois em algumas situações alterar o tempo de modificação também altera o tempo de acesso, podendo ocasionar em um pacote recentemente atualizado, mas não usado, com um peso maior do que um pacote recentemente usado.

Vale ressaltar que existe uma limitação quanto à essa abordagem, o caso de pacotes que não são diretamente executados pelo usuário. Alguns pacotes são executados toda vez que um usuário realiza certa tarefa, ou até mesmo quando o sistema é iniciado. Dessa forma, a fim de remover alguns pacotes, decidiu-se remover da lista de pacote dos usuários os pacotes com algumas prioridades definidas. A prioridade de um pacote pode ser vista também no arquivo *control* de um pacote, conforme visto na Figura 5. O Debian permite que cada pacote tenha prioridade classificada mediante os seguintes valores:

- ***Required:*** Pacotes necessários para o funcionamento básico do sistema
- ***Important:*** Pacotes esperados em qualquer distribuição Unix
- ***Standard:*** Pacotes que fornecem alguns serviços básicos esperados, como aplicações para mandar email, como o mutt.

- ***Optional:*** Pacotes que fornecem serviços que não se encaixam em nenhuma das categorias mencionadas acima.
- ***Extra:*** Pacotes que conflitam com outros de hierarquia maior ou que possuem dependências especiais que não permitem que os mesmos sejam categorizados como *Optional*.

Baseado nestas prioridades, resolveu-se por retirar do perfil do usuário qualquer pacote com prioridade *"Important" "Standard" "Required"*. Vale ressaltar que algumas linguagens de programação possuem uma das prioridades sendo filtradas, mas um processamento adicional é realizado para garantir que os pacotes relacionados a linguagens de programação continuem no perfil do usuário.

3.5 Teste de hipótese

Esta seção irá descrever como esta pesquisa pretende responder às hipóteses levantadas, mostrando assim como as duas abordagens propostas pelas hipóteses levantadas no projeto serão implementadas, assim como os resultados das mesmas serão comparadas entre si e com as abordagens já usadas no projeto selecionado.

3.5.1 Abordagem determinística

A proposta da abordagem determinística é de usar uma fórmula que altere a criação do perfil de usuário através dos pacotes instalados que foram mais recentemente usados. A Figura 8 contém o fluxo usado para realizar uma recomendação utilizando a abordagem determinística.

A abordagem determinística é a combinação da estratégia TFIDF combinada com uma fórmula para adicionar o peso do tempo ao peso calculado pelo TFIDF. Para isso acontecer, um dicionário associa cada termo, *Debtag* ou termo da descrição, aos pacotes onde os mesmos se encontram. Com esse dicionário criado, no momento que o TFIDF for calculado, o peso temporal do termo também é calculado e seus valores são enfim multiplicados.

Para calcular o peso temporal de um dado termo, calcula-se primeiro o valor da classificação no tempo para cada pacote através da fórmula definida na Equação 3.1 atribuindo o resultado da equação a cada um dos pacotes.

Com esse valor calculado, é necessário atribuir mais peso para pacotes que foram recentemente usados, sendo assim é usado uma função de decaimento exponencial para obter controle quanto à suavidade da curva que relaciona a classificação no tempo com o peso do pacote, utilizando a seguinte função:

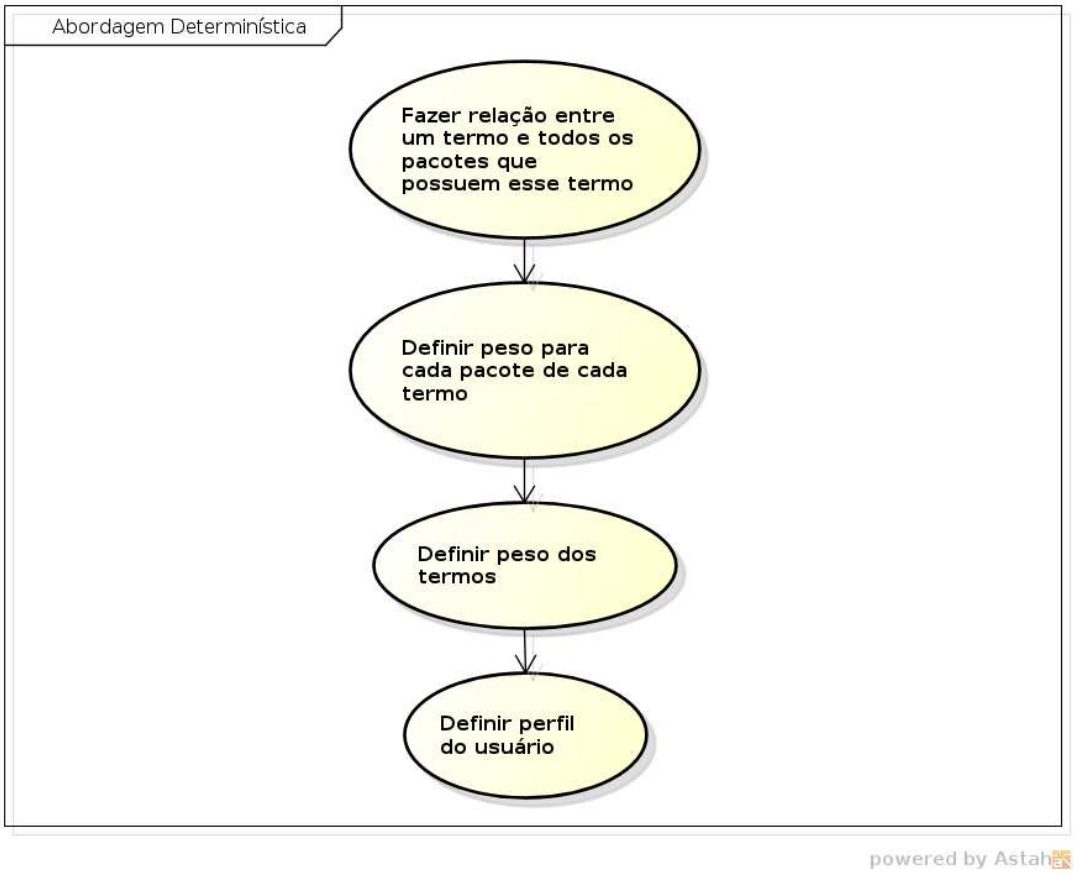

Figura 8 – Processo usado para realizar a abordagem deterministica

$$\mathbf{PP} = C \times \exp\left((1 - PTP) \times \lambda\right) \tag{3.2}$$

Onde:

- **PP:** É o peso do pacote, obtido pela curva de decaimento exponencial.
- **C e λ:** São constantes para alterar a velocidade do decaimento da curva.
- **PTP:** É o peso temporal do pacote, esse valor é obtido pela Equação 3.1.

Após calcular o peso de cada pacote do termo, o valor temporal de um termo pode ser descrito através das seguintes funções matemáticas, onde cada termo T, possui uma lista de pacotes P, onde essa lista possui um tamanho N.

$T = \{P_1, P_2, ..., P_N\}$

Além disso, cada pacote em P possui um peso w com valores entre 0 e 1. Esse peso é definido pela função W(P), que se trata do peso do pacote apresentado na Equação 3.2

$$W(Pi) = w_i, \quad 0 <= w_i <= 1$$

Aplicando essa função para cada pacote do termo é formada a lista dos pesos dos pacotes.

$$\{w_1, w_2, ..., w_n\}$$

O peso de um termo é então obtido pela média aritmética dos pesos de todos os pacotes que possuem o termo.

$$ValorTemporalDoTermo = \frac{1}{n} \times \sum_{i=1}^{n} w_i$$

O AppRecommender atribui o peso a um termo utilizando a técnica do TFIDF descrito no Apêndice A, onde o peso de cada termo está relacionado com sua frequência nos pacotes.

Com o valor temporal calculado, pode-se enfim multiplicar tal valor com o TFIDF obtido para o dado termo. Este processo é realizado para todos os termos usados para se obter o perfil do usuário.

3.5.2 Aprendizado de máquina

Um dos problemas de se usar apenas a abordagem determinística é uma limitação encontrada no *apt-xapian-index*, onde não é possível priorizar termos dentro de uma *query* quando se faz uma busca. Por exemplo, dado que o perfil do usuário possui os termos "python" e "ruby", sendo que o primeiro termo apresentou um peso maior que o segundo, na hora da criação da *query*, os pesos dos termos são ignorados. Logo, quando a busca for realizada os termos serão equivalentes, mesmo que para o perfil do usuário eles não o sejam.

Dessa forma, para a abordagem por aprendizado de máquina, escolheu-se aplicar a metodologia de filtragem pós-recomendação. Para o projeto em mãos, isso significa que o mesmo irá continuar fazendo a recomendação conforme foi desenhado, mas as recomendações serão filtradas pelo perfil de usuário construído pela abordagem de aprendizado de máquina.

Sendo assim, o enfoque dessa abordagem é usar um algoritmo supervisionado para criar o perfil do usuário baseado em seus pacotes. Dessa forma, é necessário definir como cada pacote será apresentado para algoritmo, além de definir também a categoria de cada pacote.

Para a categoria do pacote, definiu-se a seguinte escala de classificação baseada na fórmula da Equação 3.1.

Tabela 5 – Escala para classificação de um pacote

Escala	Valores
ReallyUseful(RU)	PesoTemporalDoPacote >= 0.8
Usefull(U)	PesoTemporalDoPacote >= 0.7
NotUsefull(NU)	PesoTemporalDoPacote >= 0.5

Com a escala definida e as informações de tempo coletadas para cada pacote, pode-se então criar a classificação de cada pacote manualmente instalado pelo usuário. Isso é realizado mediante um script automatizado [5] que irá realizar tal função.

Uma vez classificados, os pacotes então podem ser usados como entrada para o algoritmo supervisionado, visando assim gerar o perfil do usuário. Entretanto, ainda é necessário definir como os dados presentes no pacote serão usados de entrada para o algoritmo de aprendizado supervisionado. Considerando que os principais dados do pacote sendo usados na recomendação são as *Debtags* e os termos que compõem sua descrição, escolheu-se usar uma abordagem de vetor binário para cada um dos items. Sendo assim, o vetor de entrada será composto pela agregação de dois vetores binários.

Para as *Debtags*, o vetor binário irá ter o tamanho igual ao número de *Debtags* válidas definidas pelo AppRecommeder, sendo no caso 276 *Debtags* (ARAÚJO, 2011). Dessa forma, o vetor binário de um dado pacote irá conter o valor "0" caso o mesmo não contenha uma Debtag e "1" caso a mesma se encontre no pacote. Por exemplo, dado um conjunto de três Debtags, *role:program*, *implemented-in:c* e *devel:editor*, e um pacote com a Debtag *role:program* associado a ele, o seu vetor binário seria o seguinte: [0, 0, 1].

Para os termos da descrição, a formatação do vetor é um pouco diferente. Primeiramente, optou-se por usar os termos apenas presentes nos pacotes manualmente instalados do usuário, e não na lista de pacotes válidos usados pela aplicação. Isso se dá para reduzir o tamanho do vetor e também pelo fato de que se um termo não aparece em nenhum pacote instalado pelo usuário, o mesmo, em tese, não terá nenhum peso relevante na classificação. Além disso, usou-se a estratégia de TFIDF para retirar termos não relevantes. Para isso, calculou-se o peso TFIDF de todos os termos do usuário. Com essa informação em mãos, a média do TFIDF é encontrada e termos com valores menores que a média não são usados no vetor binário. Após aplicação deste filtro, a abordagem de vetor binário pode ser usada, funcionando de forma similar ao vetor binário das Debtags, onde os termos estarão representados em ordem alfabética e caso o pacote contenha tal termo em sua descrição, a posição do vetor associado ao termo será marcada como "1".

Com a combinação desses valores, chega-se então à forma final da entrada de um pacote para o uso de algoritmos de aprendizado supervisionados. Observa-se também que para propósitos de armazenamento, a classificação do pacote também é armazenada ao

[5] <https://github.com/TCC-AppRecommender/AppRecommender/tree/master/src/bin>

final da combinação dos vetores binários dos termos e Debtags. O papel desse vetor pode ser visto na Seção D, onde tal vetor binário representa uma linha da matrix "D" e a classificação representa uma linha da matrix "L". Logo, o conjunto de todos os pacotes que serão usados no algoritmo formam de fato a matrix "D".

Por fim, os pacotes então podem finalmente ser usados para alimentar o algoritmo de aprendizado escolhido para a pesquisa, sendo no caso um classificador bayesiano. A escolha desse classe de algoritmo se deu por alguns fatores. O primeiro se dá pela popularidade do algoritmo para sistemas de recomendação por conteúdo (AMATRIAIN et al., 2011). Outro fator que levou a escolha desse algoritmo é a sua capacidade de produzir modelos eficazes sem necessitar um enorme processamento de dados, além do fato de que o modelo gerado é facilmente compreensível, diferente de outros métodos, como redes neurais(SEGARAN, 2007).

Vale ressaltar também que algoritmos de aprendizado supervisionados não-lineares foram considerados, como redes neurais. Entretanto, segundo a pesquisa realizada por (PAZZANI; BILLSUS, 1997) o uso de redes neurais em relação ao classificador bayesiano não mostra nenhuma melhoria considerável para sistemas de recomendação baseado em conteúdo. Sendo assim, optou-se por usar o classificador bayesiano de início e caso o modelo não apresente resultados satisfatórios, mudar então para um algoritmo supervisionado não-linear.

É importante também ressaltar que o algoritmo será validado não só pela técnicas usadas no *AppRecommender*, mas também técnicas próprias de aprendizado de máquina serão usadas para verificar se o modelo gerado está satisfatório, como a validação cruzada, sendo o foco principal a identificação de *overfitting* e *underfitting*.

Com o perfil de usuário criado e validado, toda vez que uma recomendação for gerada, os pacotes escolhidos serão filtrados pelo perfil criado, sendo que apenas os pacotes classificados como "RU" serão qualificados para a recomendação ao usuário.

3.6 Comparação dos resultados

Esta seção é destinada a demonstrar como as soluções implementadas serão comparadas entre si e entre as soluções já implementadas pelo projeto selecionado. Vale ressaltar que a comparação de resultados visa principalmente responder as hipóteses levantadas na pesquisa.

Com isso em consideração, esta pesquisa escolheu seguir o modelo proposto na implementação do *AppRecommender*, entretanto com o foco nas coletas de opinião com usuários. Isso se dá pelo aspecto mais subjetivo das hipóteses da pesquisa.

Além disso, essa pesquisa tem como foco realizar duas coletas de opinião com

usuários, sendo a primeira coleta podendo ser considerada um teste piloto, onde essa coleta será realizada com um grupo de usuários, afim de encontrar problemas e observações para melhorar as soluções propostas. A segunda coleta de opinião terá foco então em propor alternativas para as questões levantadas primeira coleta, visando assim tentar melhorar seus resultados.

3.6.1 Estudo com usuários

Devido a subjetividade inerente sobre o que um usuário considera uma boa recomendação, esta pesquisa irá focar em estudos de usuários para avaliar as soluções implementadas. Para essa estratégia, será seguido um modelo similar ao usado nos testes de usuário do próprio *AppRecommender*. Entrentato algumas modificações serão realizadas no processo.

O teste de usuário realizado pelo *AppRecommender* foi realizado online, onde cada usuário tinha acesso a uma aplicação rodando o *AppRecommender*. Essa aplicação funcionava como front-end para a aplicação e escolhia uma estratégia aleatória do *AppRecommender* para ser avaliada. O usuário então teria acesso a 10 recomendações, onde cada recomendação provê uma descrição do pacote, o link para seu *upstream* e uma imagem do pacote em uso. O usuário então avaliaria o pacote recomendado e seguiria para o próximo pacote. Ao final de uma recomendação, o usuário poderia então escolher se gostaria de analisar uma outra estratégia de recomendação, onde então o processo se repetiria.

Como modificações neste processo, esta pesquisa resolveu realizar o estudo de usuário de forma offline. Além disso, o processo de teste também foi alterado. Para esta pesquisa, resolveu-se comparar uma estratégia já implementada pelo *AppRecommender* com as outras estratégias implementadas nesta pesquisa. Além disso, resolveu-se também usar uma abordagem por linha de comando, com o intuinto de agilizar os testes de usuário e possibilitar que os mesmos possam ser repetidos mais vezes.

Dessa forma, o teste de usuário contará com a recomendação de "x" pacotes por estratégia usada. Tal valor será decidido de acordo com o número de estratégias usadas. A ordem com que as recomendações são realizadas é aleatória e as informações apresentadas para cada pacote serão as mesmas definidas pelo próprio *AppRecommender*, com exceção da foto do pacote. Para cada pacote recomendado, o usuário irá poder avaliar o mesmo pelo linha de comando. Após avaliar todos os pacotes, o teste será finalizado e as informações serão coletadas pelos responsáveis pela pesquisa.

Para as métricas e a escala de avaliação, resolveu-se usar as mesmas definidas pelo AppRecommender (ARAÚJO, 2011). A escala de avaliação possui quatro itens que são definidos a seguir:

- **Ruim:** Recomendação que não agrada ao usuário.

- **Redundante:** Usuário possui aplicativos similares para o item sendo recomendado.
- **Útil:** Usuário acha que a recomendação lhe proporciona um pacote útil.
- **Surpresa boa:** Usuário considera a recomendação útil e além do mais inesperada.

Considerando isso, as métricas usadas para comparar as estratégias são as seguintes:

- **Precisão:** $\frac{NumUtil+NumSurpresaBoa}{NumRecomendacoes}$
- **Novidade:** $\frac{NumSurpresaBoa}{NumRecomendacoes}$
- **Ruim:** $\frac{NumRuim}{NumRecomendacoes}$

Onde:

- **NumUtil:** Número de recomendações marcadas como *Útil* pelo usuário.
- **NumSurpresaBoa:** Número de recomendações marcadas como *Surpresa boa* pelo usuário.
- **NumRecomendacoes:** Total de recomendações geradas para o usuário.
- **NumRuim:** Número de recomendações marcadas como *Ruim* pelo usuário.

Sendo assim, cada recomendação usada será avaliada por essas três métricas no contexto de um usuário. Ao final, as médias dessas métricas serão calculadas para todos os usuários participante da pesquisa, com o intuito de comparar as estratégias de maneira siginificativa.

4 Resultados Obtidos

Este capítulo visa relatar os resultados obtidos em ambas as coletas de opinião realizadas para verificar a validades das hipóteses propostas neste trabalho. Dessa forma, cada coleta será analisada em uma seção própria.

Vale ressaltar que ambas as coletas de dados foram realizadas com graduados e graduandos do curso de Engenharia de Software da Universidade de Brasília. Além disso, permitiu-se a coleta para qualquer usuário com sistemas Debian ou derivados do mesmo.

4.1 Primeira coleta de opinião

A primeira coleta de opinião foi realizada com 10 pessoas, onde para cada pessoa foram analisadas três estratégias de recomendação, sendo elas:

- **cbh:** Estratégia baseada em conteúdo, onde metade do perfil do usuário é formado por *Debtags* e a outra metade por termos da descrição dos pacotes. Tanto as *Debtags* e os termos são definidos utilizando a técnica do *TFIDF*;
- **cbtm:** Estratégia baseada em conteúdo utilisando o contexto temporal, assim como na estratégia 'cbh', o perfil do usuário é composto por duas metades, sendo as *Debtags* e os termos da descrição dos pacotes, porém eles são definidos através de dois pesos, sendo o primeiro uma análise de quais pacotes foram recentemente utilizados, e o segundo peso se trata da utilização do *TFIDF*;
- **cbml:** Estratégia baseada em conteúdo, onde o perfil do usuário é formado tanto por *Debtags* quanto termos, porém não é composto por metade de cada um, utilizando esse perfil é realizado a recomendação de vários pacotes. Então é utilizado o algoritmo de aprendizado de máquina para identificar quais dentre esses pacotes devem ser recomendados.

Dentre as estratégias a *cbh* já estava implementada no AppRecommender, e as estratégias *cbtm* e *cbml* se tratam das estratégias implementadas afim de testar as hipóteses desta pesquisa.

Para cada estratégia foram selecionadas as 10 melhores recomendações e essas foram ordenadas em ordem alfabética e apresentadas uma a uma para o usuário realizar a pontuação, como foi descrito anteriormente.

A coleta de dados apresentou os resultados apresentados na Figura 9

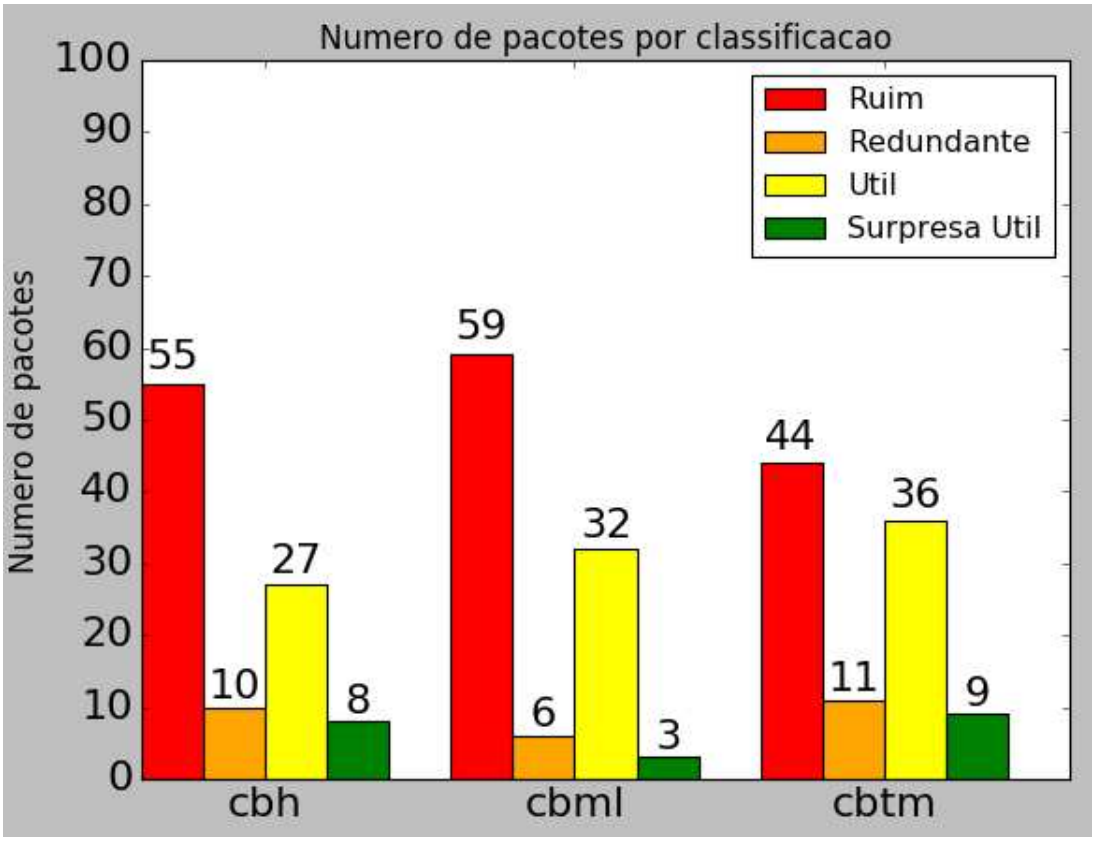

Figura 9 – Resultados da primeira coleta de opinião

Analisando os dados coletados pode-se observar que os indices de recomendações ruins das estratégias *cbh* e *cbml* foram maiores que os índices da estratégia *cbtm*, e também é notável a diferença entre recomendações úteis e surpresas boas, pois a estratégia *cbtm* possui 45 recomendações nessas classificações enquanto as estratégias *cbml* e *cbh* possuem 35.

Através dos dados coletados observa-se que mesmo a estratégia *cbtm* possuindo um pouco de destaque, todas as estratégias aprensentaram um índice muito alto de recomendações ruins

O gráfico apresentado na Figura 12 mostra a análise das métricas propostas na Seção 3.6.1, que são: precisão, novidade e ruim.

	Precisão	**Novidade**	**Ruim**
cbh	0.35	0.08	0.55
cbtm	0.35	0.03	0.59
cbml	0.45	0.09	0.44

Tabela 6 – Resultados das métricas da primeira coleta de opnião

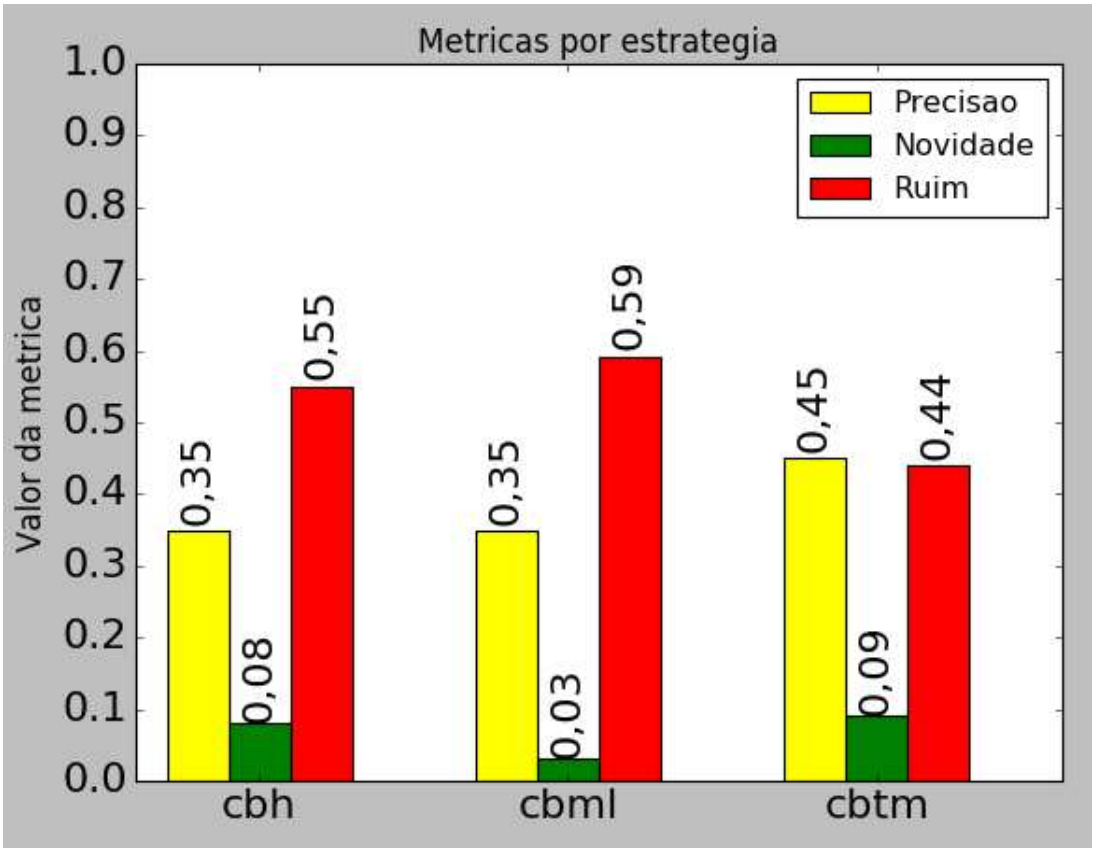

Figura 10 – Métricas da primeira coleta de opinião

Analisando a precisão e a novidade de cada recomendação, na Figura 12, observa-se que a estratégia que apresentou melhores resultados embora por pouca diferença, foi a estratégia *cbtm*, fato que pode demonstrar que a estratégia determinística se saiu melhor que a estratégia de aprendizado de máquina, para esses usuários que participaram da primeira coleta. A Tabela 6 resume os resultados das métricas encontradas:

Através dos dados coletados foi analisado a métrica de porcentagem de acertos, que é descrita na Equação 2.1, para a estratégia *cbml* através da validação cruzada, onde o resultado dessa métrica é de apenas 22,87%, resultado obtido pela média do valor apresentado por essa métrica para cada usuário, esse valor de 22,87% indica que os resultados das recomendações não são exatamente o que se espera pela técnica do cross validation, o que demonstra que até a próxima coleta de opinião deve ser realizado melhorias quanto ao tratamento dos dados, como a definição do perfil do usuário, que pode ser melhorado afim de que os resultados dessa métrica possam ser melhorados.

Os resultados dessa primeira coleta não foram o que os pesquisadores desse projeto esperavam, pois esperava-se que os resultados obtidos tivessem um resultado mais satisfatório quanto a qualidade das recomendações, porém, essa primeira coleta de opinião foi executada com o objetivo de coletar dados afim de compreender mais sobre o perfil dos usuários e o que estava sendo recomendado, e então utilizar isso para melhorar o sistema de recomendação e posteriormente realizar uma segunda coleta de opinião, onde de fato o objetivo é comparar qual estratégia se saiu melhor.

4.2 Segunda coleta de opinião

Dado os resultados da primeira coleta de opinião, as seguinte medidas foram tomadas para a segunda coleta de opinião:

- **Mudar cálculo do contexto temporal de um pacote:** Visto que pela primeira coleta, o contexto temporal do pacote não estava refletindo bem os usos do usuário. Isso percebido por analisar as informações temporais do *popularity-contest* junto com os arquivos usados pela abordagens desta pesquisa para realizar o cálculo do contexto temporal. Percebeu-se que não seria ideal apenas verificar por pacotes com arquivos binários, mas sim por qualquer pacote que forneça alguma biblioteca também. Dessa forma, o cálculo do contexto temporal de um pacote foi atualizado. O contexto temporal de um pacote agora é calculado de forma semelhante ao obtido pelo *popularity-contest*, onde o tempo de acesso de cada arquivo relacionado a um pacote é observado e o maior tempo encontrado é considerado o tempo de acesso.

- **Balancear rótulo dos pacotes:** Com o intuito de melhorar a acurácia dos algoritmos de aprendizado de máquina, resolveu-se melhor balancear os pacotes do usuário. Para isso, os pacotes são ordenados pelo seu contexto temporal e divididos em três grupos de igual tamanho, onde cada grupo representa um rótulo possível.

- **Selecionar pacotes apenas instalados diretamente pelo *apt*:** Para melhorar o perfil de usuário, resolveu-se selecionar não mais os pacotes marcados como manualmente instalados pelo sistema, e sim apenas aqueles cujo usuário instalou diretamente pelo *apt*.

- **Implementar nova forma de representar um pacote:** Para verificar se outras formas de representação de um pacote podem ser mais eficientes, implementou-se uma nova estratégia de aprendizado de máquina, que usa o modelo *Bag Of Words* juntamente com o *TFIDF* para representar um pacote.

- **Uso da estratégia de expansão de query:** Essa estratégia já implementada pelo *AppRecommender* faz com que o *Xapian* apenas receba os pacotes que deva fazer a

pesquisa relacionada e o mesmo deve então ser capaz de retirar de cada pacote as informações mais importantes para realizar a pesquisa. Dessa forma, as estratégias de aprendizado de máquina também foram combinadas com essa estratégia com o intuito de verificar se melhores resultados foram gerados.

- **Diminuir número de pacotes passados para o Xapian:** Para as estratégias de aprendizado de máquina, apenas os pacotes marcados como *Really Useful* (RU) são usados para criar o perfil do usuário. Isso se dá pelo fato de tentar reduzir o perfil do usuário para que o mesmo seja mais focado nos gostos atuais do usuário.

- **Uso da estratégia *cb* ao invés da *cbh*:** Para a primeira coleta de opinião, a estratégia *cbh* foi usada. Entretanto, a segunda coleta quis colocar mais foco nos termos dos pacotes e não de forma equilibrada como funciona o *cbh*. Dessa forma, a estratégia do *AppRecommender* usada para comparação com as estratégias de contexto temporal, foi a estratégia *cb*, que usa tanto termos do pacote como *Debtags*, mas não garante que as mesmas apareçam na mesma quantidade no perfil do usuário.

- **Mudança de nome da estratégia *cbml* para *mlbva*:** Como agora existem duas estratégias distintas usando aprendizado de máquina, resolveu-se renomear a estratégia *cbml* para *mlbva*, que significa *Machine Learning Binary Vector Approach*. Esse padrão também é seguido para a estratégia de *Bag Of Words*, *mlbow*, que significa *Machine Learning Bag Of Words*.

- **Retirar pacotes que possuem dependências de programas não instalados no sistema do usuário:** Durante a primeira coleta de opinião, foi observado a recomendação de alguns pacotes relacionados a pacotes que não estavam presentes no sistema do usuário, como o pacote *auto-complete-el*, que é um pacote que adiciona a funcionalida de auto-completar para o pacote *emacs*. Entretanto, se o usuário não tem o pacote *emacs* instalado, tal recomendação não é de grande ajuda. Dessa forma, pacotes que dependem de pacotes executáveis, só podem ser recomendados se tais dependências já estiverem instaladas no sistema do usuário.

Além desses pontos já citados, a segunda coleta de opinião também foi realizada com mais usuários, 24 no total, sendo que os usuários escolhidos também apresentavam um perfil mais geral quanto aos sistemas operacionais usados, pois para a primeira coleta os usuários eram majoritariamente usuários Debian.

Dito, isso o resultado geral da pesquisa pode ser visto no Gráfico 11, onde seus *labels* são respectivamente:

- **mlbow**: Aprendizado de máquina com *Bag Of Words*

- **mlbva_eset**: Aprendizado de máquina com vetor binário e expansão de query.

- **cb**: Estratégia com perfil de usuário sem contexto temporal.
- **cbtm**: Estratégia deterministica com contexto temporal.
- **mlbva:** Aprendizado de máquina com vetor binário.
- **mlbow_eset:** Aprendizado de máquina com *Bag Of Words* usando expansão de query.
- **cb_eset:** Estratégia usando apenas expansão de query, sem contexto temporal.

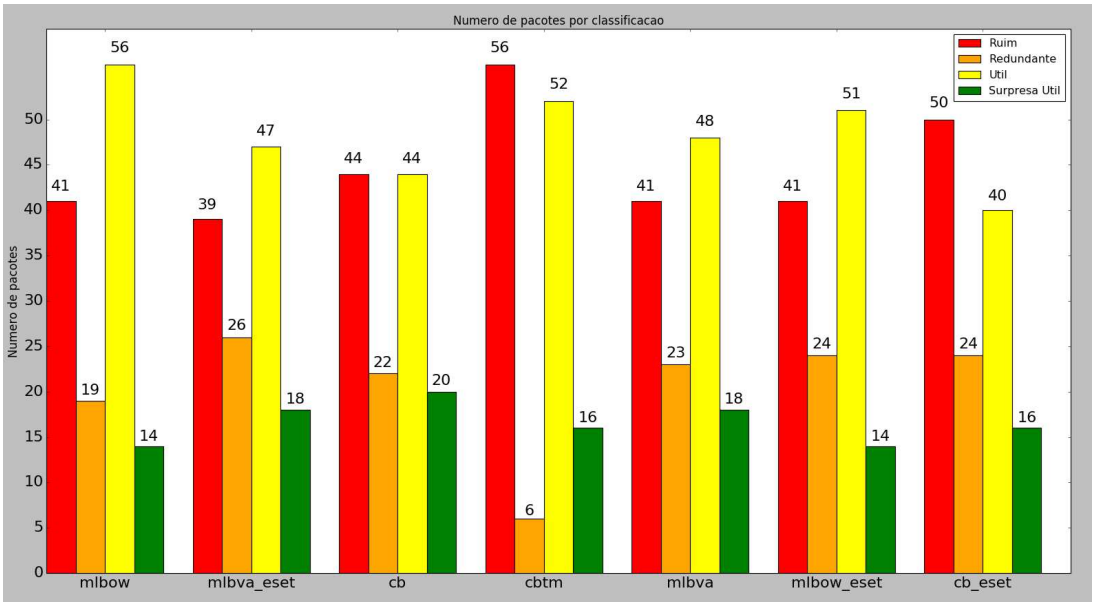

Figura 11 – Resultados da segunda coleta de opinião

Visualmente, pode-se perceber que apenas as estratégias de aprendizado de máquina apresentaram um total de recomendações úteis ao usuário maior que o total de recomendações ruins. Além disso, percebe-se que as estratégias como um todo estão produzindo poucos resultados considerados uma surpresa para usuário e estão gerando números próximos de pacotes redundantes.

Uma melhor análise pode ser feita ao observar o Gráfico 12 com a relação as métricas calculadas para cada estratégia, conforme definidas na Seção 3.6.1

Pode-se ver pelo Gráfico 12 quanto a precisão, o melhor resultado foi para estratégia *mlbow*. Entretanto, a melhora se deu em apenas 5% em relação as estratégias baseadas em conteúdo do *AppRecommender*. Vale ressaltar também que a métrica **novidade** mostra o que foi observado no Gráfico 11, onde o número de recomendações marcadas como surpresa se manteve estável entre as ferramentas.

Além disso, pode-se perceber que a estratégia determinística com contexto temporal *cbtm* foi a que recomendou mais pacotes considerados ruins pelo usuário, ou seja,

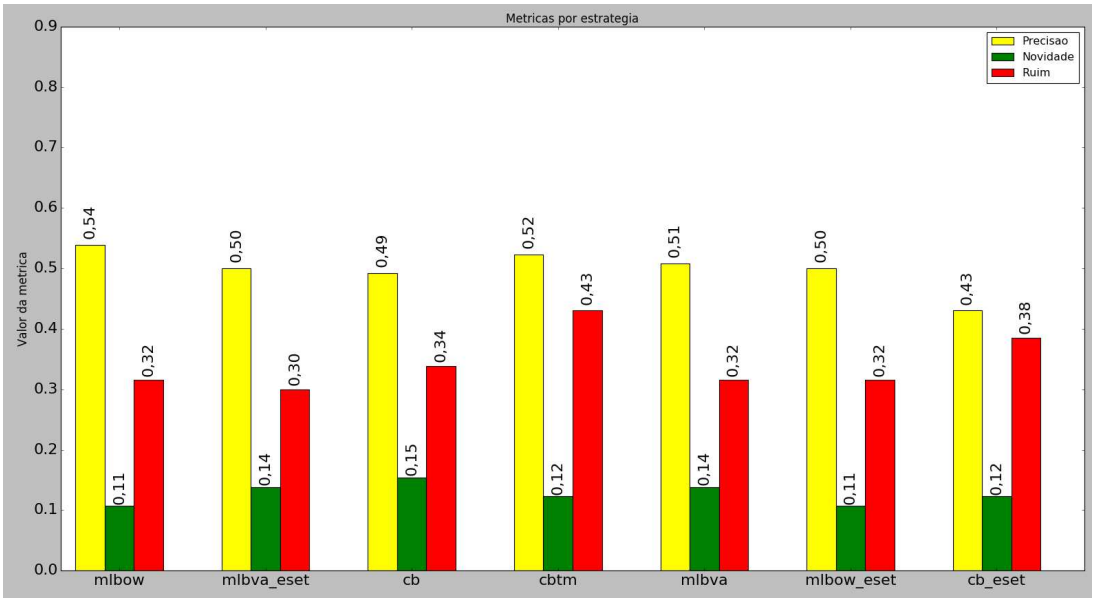

Figura 12 – Métricas da segunda coleta de opinião

a estratégia não atende aos requisitos. Isso difere dos resultados encontrados na primeira coleta de opinião. Percebeu-se então que a mudança no cálculo temporal favoreceu as estratégias de aprendizado de máquina, mas não a determinística.

Sobre a questão de redundância, percebeu-se um problema quanto aos pacotes instalados por outros gerenciadores de pacote. Por exemplo, caso o usuário seja um programador ruby e tenha instalado pacotes ruby via *apt* e *Ruby Version Manager* (rvm), o *AppRecommender* pode recomendar pacote instalados pelo *rvm*, o que causa a redundância.

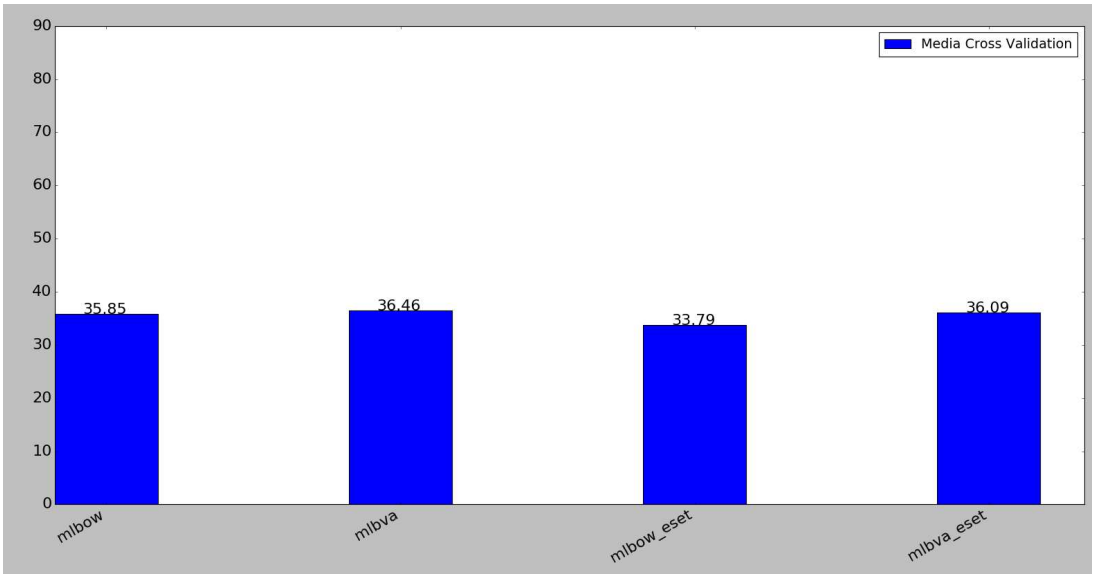

Figura 13 – Métrica de porcentagem de acertos obtida via cross validation

Por fim, é necessário entender por que a melhoria encontrada foi de 3%-5% em

	Precisão	**Novidade**	**Ruim**
mlbow	0.54	0.11	0.32
mlbva_eset	0.50	0.14	0.30
cb	0.49	0.15	0.30
cbtm	0.52	0.12	0.43
mlbva	0.51	0.14	0.32
mlbow_eset	0.50	0.11	0.32
cb_eset	0.43	0.12	0.38

Tabela 7 – Resultados das métricas da segunda coleta de opnião

relação as estratégias baseadas em conteúdo já implementadas pelo *AppRecommender*. O gráfico da Figura 13 mostra a média da métrica de acurácia da porcentagem de acertos, definida na Equação 2.1, para cada estratégia de aprendizado de máquina. Tal gráfico mostra que nenhuma estratégia teve uma média muito alta de acurácia, sendo que acredita-se que a melhora encontrada por essas estratégias foi devido a pré-filtragem de pacotes e não para a pós filtragem dos mesmos. Além disso, a Tabela 7 resume os resultados encontrados no segundo experimento.

Vale ressaltar que mesmo com a estratégia de pré-filtragem sendo a que tenha proporcionado a melhora nos resultados, pode-se ver que entre as estratégias de aprendizado de máquina, as estratégias de classificação também tiveram influência na melhoria dos resultados, pois todas as estratégias faziam a pré-filtragem de forma idêntica, mas foi a *mlbow* com o uso do *Bag Of Words* e do *TFIDF* que apresentou os melhores resultados.

À partir dessa análise, acredita-se que um dos principais motivos para a baixa acurácia dos algoritmos se dá pelo número de dados usado pelo treinamento. Em média, apenas 37 pacotes estavam sendo usados para treinar o algoritmo, o que é uma quantidade baixa para o treinamento do algoritmo propriamente dito.

Além disso, o método de aprendizado de máquina utilizado, o Bayes ingênuo pode ser um método muito simples para entender a questão do gosto do usuário e seus pacotes, sendo que outros métodos poderiam ter sido explorados.

5 Conclusão

Com os resultados obtidos, pode-se perceber que ambas as hipóteses, *h1* e *h2* não podem ser negadas. Para hipótese *h1*, que se trata da recomendação ser mais apropriada ao aumentar o peso dos pacotes recentemente usados no sistema, percebeu-se um aumento em todas as estratégias de contexto temporal quanto a precisão das recomendações, entretanto, a estratégia determinística também teve um aumento de recomendações negativas ao usuário, o que leva-se a entender que o contexto temporal sozinho não foi responsável pelas melhorias. Para a hipótese *h2*, que dita que a estratégia de aprendizado de máquina obtém melhores resultados que a estratégia determinística, percebeu-se que sim, as estratégias de aprendizado de máquina se comportaram melhor que a determinística. Acredita-se que esse foi o caso devido a combinação de pré-filtragem dos pacotes juntamente com o contexto temporal que acontecia nas estratégias de aprendizado de máquina.

Com a melhoria encontrada nos resultados obtidos, pode-se entender também a importância de uma melhor filtragem do perfil de usuário antes da recomendação em si. Sendo assim, foi entendido que o contexto temporal proposto não precisa estar associado apenas a estratégia de recomendação em si, mas também a todas as etapas da recomendação. Isso se mostrou pertinente neste trabalho, pois o contexto temporal foi aplicado tanto na pós-filtragem dos pacotes, quanto na composiçãod dos pacotes no perfil do usuário.

Com esta análise dos resultados, entendeu-se também que o foco da recomendação não deve estar apenas em qual estratégia utilizar para fazer a recomendação, e sim na seleção de quais pacotes devem alimentar o recomendador e depois usar uma estratégia para representar e selecionar os pacotes para uma recomendação.

Além disso, a pesquisa também permitiu entender que algumas contribuições podem ser feitas ao próprio sistema Debian para melhorar ainda mais os resultados de uma recomendação. A principal delas seria a criação de um log para o uso de pacotes, similar ao que acontece com o *relatime*, onde diariamente, seria registrado quais pacotes tiveram seu *atime* alterado. Esse arquivo iria conter a frequência de uso dos pacotes pelo usuário, provendo essa frequência de uso dos pacotes ao contexto temporal, permitindo uma análise ainda mais próxima do perfil do usuário.

Ainda se tratando dos resultados encontrados, verificando a média de acurácia do *cross validation* e o número de dados médio para o treinamento dos algoritmos, pode-se entender que uma abordagem de aprendizado de máquina que consiga resultados mais expressivos de acurácia e que se comporte melhor para menores conjuntos de dados, pode melhorar ainda mais os resultados encontrados.

Por fim, vale ressaltar que este trabalho possibilitou que ambos os pesquisadores envolvidos trabalhassem no programa *Google Summer of Code*, onde a instituição *Google* financia estudantes para trabalharem com software livre durante o verão. Os pesquisadores deste trabalho estão trabalhando também com o *AppRecommender*, porém em outras áreas não relacionadas a esta pesquisa. Dessa forma, pode-se dizer que além dos resultados obtidos, esta pesquisa foi capaz de reviver o *AppRecommender* dentro do projeto *Debian*, e também pretende atrair mais pesquisadores para dar continuidade ao projeto.

5.1 Limitações do trabalho

Uma das principais limitações deste trabalho se encontra no aspecto subjetivo de uma avaliação de uma recomendação e no grupo de usuários que realizaram a coleta de opinião. Considerando que apenas graduados e graduandos do curso de Engenharia de Software da Universidade de Brasília foram consultados, pode existir talvez um perfil comum entre os participantes no que tange os resultados encontrados.

De forma a melhor verificar os resultados encontrados nesta pesquisa, decidiu-se por levantar a hipótese de realizar uma coleta de opinião em outros dois eventos de software livre, a *DebConf*, conferência anual do projeto Debian, e o Fórum Internacional De Software Livre (FISL), evento anual de encontro de comunidades de software livre. Com as consultas realizadas nesses dois eventos, pretende-se então construir um perfil de usuário mais heterogêneo, permitindo assim uma melhor análise das estratégias desenvolvidas.

5.2 Trabalhos futuros

Dado as questões levantadas pela pesquisa, foi possível levantar alguns possíveis trabalhos futuros:

- **Uso do arquivo history do usuário para complementar contexto temporal:** Apesar do arquivo *bash_history* do usuário apenas conter um histórico de comandos usados pelo usuário, tal arquivo pode conter informações importantes sobre a frequência de uso de algumas aplicações pelo usuário, podendo complementar a análise temporal feita.
- **Melhorar estratégias de aprendizado de máquina:** Dado os problemas já citados pelas estratégias de aprendizado de máquina, pode-se investigar novas metodologias para lidar com os problemas citados, como poucos dados de treinamento e complexidade de uso dos mesmos.

- ***Cluster* de interesse do usuário:** Essa nova estratégia seria para criar *clusters* de gostos do usuário com base em seus pacotes e buscar pacotes baseado na identidade desse *cluster* e não nos pacotes em si.

- **Melhorar contexto temporal:** As estratégias de contexto temporal ainda apresentam problemas para pacotes recentemente atualizados. Isso se dá pelo fato que ainda não é possível saber se um pacote foi recentemente utilizado e atualizado pelo sistema, pois a operação de atualização opera tanto sobre o *ctime* quanto *atime*. Dessa forma, pode-se buscar maneira de resolver tal problema.

- **Evitar redundância de recomendação de pacotes:** Pacotes instalados pelo usuário por outros gerenciadores de pacotes, como *rvm* do ruby ou o *pip* do python, não devem ser recomendados. Dessa forma, deve-se entender um mecanismo de verificar a existência desses pacotes e retirá-los da recomendação. Além disso, deve-se verificar também redundância entre pacotes dentro de uma recomendação, pois dois pacotes similares não devem estar presentes na mesma recomendação.

- **Uso de experimentos *offline*:** Experimentos *offline* eram implementados no *AppRecommender* com o intuito de comparar as estratégias de forma mais rápida e prática, pois não era necessário sempre fazer testes com usuários. Para o contexto temporal usado, tais experimentos deveriam ser modificados e melhor explorados para se adequar a finalidade da pesquisa.

Referências

ADOMAVICIUS, G.; TUZHILIN, A. Context-aware recommender systems. In: *Recommender systems handbook*. [S.l.]: Springer, 2011. p. 217–253. Citado na página 17.

AMATRIAIN, X. et al. Data mining methods for recommender systems. In: *Recommender Systems Handbook*. [S.l.]: Springer, 2011. p. 39–71. Citado 2 vezes nas páginas 21 e 39.

ARAÚJO, T. C. *AppRecommender: um recomendador de aplicativos GNU/Linux*. Tese (Doutorado) — Universidade de Sao Paulo, 2011. Citado 9 vezes nas páginas 11, 16, 22, 25, 27, 31, 38, 40 e 58.

BASILE, P. et al. Modeling short-term preferences in time-aware recommender systems. *Proceedings of DeCAT*, 2015. Citado 2 vezes nas páginas 23 e 24.

BERRY, M. J.; LINOFF, G. *Data mining techniques: for marketing, sales, and customer support*. [S.l.]: John Wiley & Sons, Inc., 1997. Citado na página 17.

BURKE, R. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, Springer, v. 12, n. 4, p. 331–370, 2002. Citado na página 11.

BURKE, R. *Hybrid web recommender systems, The adaptive web: methods and strategies of web personalization*. [S.l.]: Springer-Verlag, Berlin, Heidelberg, 2007. Citado na página 15.

CORBET, J. *Once upon atime [LWN.net]*. 2007. Disponível em: <http://lwn.net/Articles/244829/>. Citado na página 34.

CUSTóDIO, B.; HENRIQUE, D. Aprendizagem de máquina. 2010. Disponível em: <http://www.ft.unicamp.br/liag/wp/monografias/monografias/2010_IA_FT_UNICAMP_aprendizagemMaquina.pdf>. Citado na página 18.

DING, Y.; LI, X. Time weight collaborative filtering. In: ACM. *Proceedings of the 14th ACM international conference on Information and knowledge management*. [S.l.], 2005. p. 485–492. Citado na página 23.

HAAS, J. *stat - Linux Command - Unix Command*. Disponível em: <http://linux.about.com/library/cmd/blcmdl2_stat.htm>. Citado na página 33.

JANNACH, D. Finding preferred query relaxations in content-based recommenders. In: *Intelligent Techniques and Tools for Novel System Architectures*. [S.l.]: Springer, 2008. p. 81–97. Citado na página 11.

KOREN, Y. Collaborative filtering with temporal dynamics. *Communications of the ACM*, ACM, v. 53, n. 4, p. 89–97, 2010. Citado na página 23.

KOREN, Y.; BELL, R.; VOLINSKY, C. Matrix factorization techniques for recommender systems. *Computer*, IEEE, n. 8, p. 30–37, 2009. Citado na página 11.

LECUN, Y. et al. Backpropagation applied to handwritten zip code recognition. *Neural computation*, MIT Press, v. 1, n. 4, p. 541–551, 1989. Citado na página 18.

LEE, T. Q.; PARK, Y.; PARK, Y.-T. A time-based approach to effective recommender systems using implicit feedback. *Expert systems with applications*, Elsevier, v. 34, n. 4, p. 3055–3062, 2008. Citado na página 23.

LOPS, P.; GEMMIS, M. D.; SEMERARO, G. Content-based recommender systems: State of the art and trends. In: *Recommender systems handbook*. [S.l.]: Springer, 2011. p. 73–105. Citado 2 vezes nas páginas 15 e 16.

MAHMOOD, T.; RICCI, F. Improving recommender systems with adaptive conversational strategies. In: ACM. *Proceedings of the 20th ACM conference on Hypertext and hypermedia*. [S.l.], 2009. p. 73–82. Citado na página 14.

PAZZANI, M.; BILLSUS, D. Learning and revising user profiles: The identification of interesting web sites. *Machine learning*, Springer, v. 27, n. 3, p. 313–331, 1997. Citado na página 39.

PICAULT, J. et al. How to get the recommender out of the lab? In: *Recommender Systems Handbook*. [S.l.]: Springer, 2011. p. 333–365. Citado 2 vezes nas páginas 14 e 15.

RAJARAMAN, A.; ULLMAN, J. D. *Mining of Massive Datasets*. [S.l.]: Cambridge University Press, 2011. Citado na página 57.

RICCI, F.; ROKACH, L.; SHAPIRA, B. *Introduction to recommender systems handbook*. [S.l.]: Springer, 2011. Citado 2 vezes nas páginas 14 e 16.

ROBERTO, P. et al. Utilização de técnicas de aprendizado de máquina no reconhecimento de entidades nomeadas no português. 2011. Disponível em: <https://www.google.com.br/url?sa=t&rct=j&q=&esrc=s&source=web&cd=5&ved=0CDoQFjAEahUKEwjVzP-CqYHJAhWIIpAKH%3A%2F%2Frevistas.unibh.br%2Findex.php%2Fdcet%2Farticle%2Fdownload%2F305%2F164&usg=AFQjCNFRRA7rednn_zxKwb-EosOQK5nAOg&sig2=c7f9GVYNNDBLsHpkwCA6Vg&bvm=bv.106923889,d.Y2I&cad=rja>. Citado 2 vezes nas páginas 18 e 19.

SCHAFER, J. B. et al. Collaborative filtering recommender systems. In: *The adaptive web*. [S.l.]: Springer, 2007. p. 291–324. Citado na página 16.

SEGARAN, T. *Programming collective intelligence: building smart web 2.0 applications*. [S.l.]: "O'Reilly Media, Inc.", 2007. Citado 2 vezes nas páginas 20 e 39.

SHANI, G.; GUNAWARDANA, A. Evaluating recommendation systems. In: *Recommender systems handbook*. [S.l.]: Springer, 2011. p. 257–297. Citado na página 18.

SILVA, E. L. da; MENEZES, E. M. Metodologia da pesquisa e elaboração de dissertação. *UFSC, Florianópolis, 4a. edição*, 2005. Citado na página 23.

STALLKAMP, J. et al. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural networks*, Elsevier, v. 32, p. 323–332, 2012. Citado na página 18.

ZHANG, H. The optimality of naive bayes. *AA*, v. 1, n. 2, p. 3, 2004. Citado na página 20.

ZINI, E. A cute introduction to debtags. In: *Proceedings of the 5th annual Debian Conference*. [S.l.: s.n.], 2005. p. 59–74. Citado na página 30.

Apêndices

APÊNDICE A – Term Frequency-Inverse Document Frequency

A técnica *Term Frequency-Inverse Document Frequency (TFIDF)* é uma medida estátistica usada prioritariamente para indicar a importância de um termo em dado documento em relação a uma base de documentos (RAJARAMAN; ULLMAN, 2011).

Para calcular tal fator, duas etapas distintas são necessárias:

- ***TF: Term Frequency:*** Calcula quantas vezes um termo aparece em um dado documento. Dessa forma, seu cálculo pode realizado pela seguinte fórmula:

  TF(t) = N(t,d)/T(d)

  Onde:

  - **N(t, d):** Número de vezes que o termo "t" aparece no documento "d".
  - **T(D):** Número de termos presentes no documento "d".

- ***IDF: Inverse Document Frequency:*** Usada para diminuir o peso de termos muito frequentes em documentos, como os termos "para" ou "é". Para fazer isso, a seguinte fórmula pode ser usada:

  IDF(t) = log(NumDocumentos/D(t))

  Onde:

  - **NumDocumentos:** Quantidade de documentos presentes na base.
  - **D(T):** Número de documentos que contém o termo "t".

Com os fatores **TF** e **IDF** calculados, basta multiplicar tais valores para encontar o valor da importância de um termo dado uma base de documentos. Sendo assim, pode-se entender que o valor do *TFIDF* será alto para um termo "t" se o mesmo é encontrado muitas vezes em um dado documento e poucas vezes na coleção de documentos.

Existem também outras formas de calcular os valores *TF* e *IDF*. Essas formas tendem a priorizar certos aspectos em relação ao texto. Uma dessas abordagens é chamada de *TFIDF-sublinear*. Essa técnica é usada para atenuar o valor do *TF* para quando um termo é bastante presente em um documento, mas não é muito presente na base de documentos. A atenuação do valor do *TF* pode se justificar em casos quando um termo que aparece mais de uma vez no documento não deva ter um peso substancialmente maior

que um termo que aparece uma única vez. Para calcular o *TF* para este caso, pode-se usar a seguinte fórmula:

$$\text{TF-sublinear(t)} = 1 + \log(\text{TF(t)})$$

Vale ressaltar que outras abordagens também podem ser usadas, como normalização usando o tamanho médio dos documentos presentes na base de todos os documentos e *TF* médio entre todos os *TF*s (ARAÚJO, 2011).

APÊNDICE B – Coleta de dados do usuário

A coleta de dados do usuário é realizada através do script “collect_user_data.py”, localizado no caminho “src/bin/data_collect” dentro do projeto AppRecommender. A execução do script segue os seguintes passos:

- ***Coletar informações do sistema:*** Nesse passo é coletado as informações quanto a versão do kernel do linux, e a versão da distribuição, essas informações são coletadas através dos seguintes comandos:
  - Versão do kernel: $ uname -s -r
  - Versão da distribuição: $ lsb_release -a

  Essas informações são armazenadas no arquivo “pc_informations.txt”;
- ***Coletar todos os pacotes do usuário:*** Esses pacotes são obtidos através do comando “$ /usr/bin/dpkg –get-selections”, que retorna uma lista de pacotes do usuário onde para cada pacote segue a informação “install” ou “deinstall”, então é feito um filtro para capturar apenas os pacotes instalados, ou seja, que possuem a informação “install”. Então essa lista de pacotes é armazenada no arquivo “all_pkgs.txt”;
- ***Coletar pacotes manualmente instalados:*** Esses pacotes são coletados através do comando “$ apt-mark showmanual” que retorna a lista de pacotes manualmente instalados, e após coletar os pacotes eles são armazenados no arquivo “manual_installed_pkgs.txt”;
- ***Coletar tempo dos pacotes:*** Nesse passo é coletado o tempo de acesso e de modificação de cada pacote. Estes dados serão usados afim de obter informação quanto a classificação do pacote em relação ao tempo que este foi utilizado, para isso se coleta o tempo de acesso e o tempo de modificação de cada pacote, esses tempos são coletados através do comando “stat”, onde se utiliza a flag “-c” com o parâmetro “%Y” para coletar o tempo de modificação, ou o parâmetro “%X” para se coletar o tempo de acesso, e após esse parâmetro é passado o nome do executável do pacote desejado, para otimizar a busca do pacote é utilizado o comando “which”, que retorna o caminho completo para o executável. Logo os dados do tempo de acesso e de modificação são coletados através do seguinte comando:
  - Tempo de acesso: $ stat -c ’%X’ ‘which [executável]‘
  - Tempo de modificação: $ stat -c ’%Y’ ‘which [executável]‘

Esses dados são coletados e então armazenados no arquivo "pkgs_time.txt", onde para cada linha há o nome do pacote seguido do tempo de modificação e depois o tempo de acesso;

- ***Caminho do binário de cada pacote:*** Esses dados são coletados através do comando "which", passando como parâmetro o nome do pacote, ou seja, é executado o comando "$ which [pacote]" para cada um dos pacotes do usuário e então esses dados são armazenados no arquivo "pkgs_binary.txt", onde em cada linha há o nome do pacote seguido do caminho para o seu binário;

- ***Submissão do popularity-contest:*** Os dados do popularity-contest são obtidos através da execução do script "popularity-contest", que está no mesmo diretório do script de coleta de dados. Vale ressaltar que esse é o script original do popularity contest. Essa abordagem foi escolhida pois para executar o popularity contest, permissão de super usuário era necessária, o que pode ocasionar problemas na coleta de dados. Dessa forma, adaptou-se o script original, modificando por exemplo o caminho de onde o mesmo buscava o arquivo de configuração necessário e também criando a própria versão do arquivo de configuração para a pesquisa.

Todas essas coletas citadas acima são executadas em uma thread, enquanto na thread principal é executado a coleta de preferências do usuário, onde é executado o AppRecommender com algumas estratégias, dentre elas as criadas neste trabalho. Após a execução dessas estratégias de recomendação os pacotes recomendados, de todas as estratégias, são colocados em uma única lista, e então ordenados pelo nome, em seguida é passado pacote por pacote solicitando ao usuário que faça uma avaliação do pacote, pontuando cada um deles de 1 a 4. Após o usuário terminar de avaliar os pacotes é criado um arquivo para cada recomendação, onde cada arquivo inicia com a sigla da recomendação seguido de "_recommendation.txt".

O diretório onde os arquivos da coleta são armazenados é o "app_recommender_log/", localizado na raiz do sistema do usuário.

APÊNDICE C – Bag of Words

O modelo *Bag of Words* é uma possível representação vetorial de um dado conjunto de texto. Este modelo ordena o texto em um conjunto de palavras de acordo com sua ordem alfabética, e então conta o número de vezes que uma palavra aparece no texto. Por examplo, dado o texto:

"The only good way to become good at programming is programming"

Ao ordenar esta frase pelo seus termos, têm-se:

"at become good good is only programming programming the to way"

Vale ressaltar que conforme o exemplo acima, uma normalização foi realizada, onde o texto é colocado todo em letras minúsculas.

Com o texto ordenado, o vetor então é criado. Cada posição desse vetor simboliza uma palavra do texto ordenado. O valor de cada posição, simboliza o número de vezes que uma palavra aparece no texto. Dessa forma, para o exemplo já demonstrado, têm-se o seguinte vetor:

$$BagOfWords = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 1 \\ 1 \\ 2 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Pode-se perceber que apenas as palavras que apareceram duas vezes são marcadas com valor "2" no vetor. Tal representação pode ser combinada com o *TFIDF* para eliminar termos muito recorrentes em um texto e deixar a representação dos vetores menores.

Apesar da representação simples, tal modelo não leva em consideração a ordem das palavras e muito menos a relação entre elas, o que dependendo do problema a ser resolvido, pode-se tornar uma representação muito simples de um conjunto de textos.

APÊNDICE D – Bayes Ingênuo

Dado o contexto do classificador bayesiano apresentado na Seção 2.2.2, resolveu-se implementar tais fórmulas usando apenas manipulação de matrizes. Essa decisão se deu principalmente por questões de performance, principalmente pelo fato de que o perfil do usuário pode mudar com alta frequência, necessitando que o treinamento do algoritmo seja eficaz.

Para mostrar o modelo criado, a Tabela 1 será usada. Primeiramente, esta tabela será dividida em duas matrizes distintas, D e L. A matriz D representa os atributos de cada item. Dessa forma, sua dimensão será "$e \times a$", onde "e" é significa os número de itens na tabela, e "a" representa o número de atributos que cada item possui. Já a matriz L representa a classificação para cada item "e" encontrado na matriz D. Dessa forma, a mesma será uma matriz coluna com dimensão "$e \times 1$".

$$D = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}_{e \times a}$$

$$L = \begin{bmatrix} 1 \\ 2 \\ 0 \\ 1 \\ 2 \end{bmatrix}_{e \times 1}$$

Para o algoritmo funcionar também é necessário ter conhecimento de quais são as classificações possíveis, essas classificações são informadas através do vetor coluna B, com dimensões "$l \times 1$", onde "l" é o número de classificações possíveis.

$$B = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}_{l \times 1}$$

Utilizando a matriz D e os vetores L e B, monta-se a matriz de adjacência A, onde cada linha dessa matriz representa respectivamente uma classificação possível, e cada coluna da matriz identifica se um item da matriz D foi classificado de acordo com a classificação representada pela linha em que se encontra. A matriz de adjacência A também

é uma matriz com valores binários, onde "1" indica a presença do item na classificação, e "0" indica a ausência do item na classificação. Logo a matriz A possui dimensões "$l \times e$".

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix}_{l \times e}$$

Utilizando a matriz A pode-se calcular o vetor com o histograma do vetor L, onde cada linha do vetor histograma H representa uma possível classificação da matriz B, e o valor de cada coluna representa o número de itens no qual a classificação está associada. O vetor coluna H é calculado pela multiplicação da matriz A pelo vetor coluna com todos os valores sendo "1", onde esse vetor coluna possui "e" linhas, logo as dimensões do vetor coluna H são "$l \times 1$"

$$H = A_{l \times e} * \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}_{e \times 1}$$

$$H = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}_{l \times 1}$$

Utilizando o vetor H, é calculado a probabilidade de cada classificação, que é representada pelo vetor PH, resultante da divisão do vetor H por "e".

$$PH = \frac{1}{e} * H_{l \times 1}$$

$$PH = \begin{bmatrix} \frac{1}{5} \\ \frac{2}{5} \\ \frac{2}{5} \end{bmatrix}_{l \times 1}$$

O próximo passo é calcular a matriz PR1, que se trata da probabilidade de um atributo ser igual a "1", na relação entre as classificações possíveis, o vetor B, e os atributos da matriz D. Para isso é necessário identificar quantos atributos estão presentes em cada uma das classificações possíveis, valor esse obtido pela multiplicação da matriz A com a matriz D, resultando na matriz R contendo para cada possível classificação quantos atributos estão presentes na mesma. Com a matriz R calculada, pode-se obter a matriz de probabilidade PR1, que é o resultado da multiplicação entre a matriz inversa da diagonal feita pelo vetor H com a matriz R.

$$R = A_{l\times e} * D_{e\times a}$$

$$R = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 2 & 0 & 2 & 1 & 1 \\ 0 & 2 & 2 & 1 & 1 \end{bmatrix}_{l\times a}$$

$$diag(H) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}_{l\times l}$$

$$diag(H)^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} \end{bmatrix}_{l\times l}$$

$$PR1 = diag(H)^{-1}_{l\times l} * R_{l\times a}$$

$$PR1 = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & \frac{1}{2} & \frac{1}{2} \\ 0 & 1 & 1 & \frac{1}{2} & \frac{1}{2} \end{bmatrix}_{l\times a}$$

Também é necessário identificar a matriz PR0, que indica a probabilidade dos atributos possuirem o valor zero, essa matriz pode ser obtida através da expressão "$1 - PR1$".

$$PR0 \ = \ 1 \ - \ PR1_{l\times a}$$

$$PR0 = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & \frac{1}{2} & \frac{1}{2} \\ 1 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix}_{l\times a}$$

Obtendo o vetor PH, que é a probabilidade individual de cada rótulo, e a matrizes PR1 e PR0, o treinamento do algoritmo está completo.

Com o algoritmo treinado, ele está pronto para receber um vetor com os atributos, como mostra a Tabela 2, é usado como exemplo o vetor v, que possui dimensões "$1 \times a$".

$$v = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \end{bmatrix}_{1\times a}$$

O próximo passo é montar a matriz PV, que se trata da matriz de probabilidade para o vetor. Essa matriz é criada através da junção das matrizes PR1 e PR0. Entretanto,

esse junção tem relação direta com os valores do vetor v, pois se uma de suas colunas encontra-se o "1", deve-se utilizar a coluna da matriz PR1, e o contrário, utilliza-se a coluna da matriz PR0. Em outras palavras, a matriz PV é resultante da soma dos produtos entre PR1 e a matriz diagonal de v, com PR0 e a matriz diagonal de v', onde o vetor v' se trata do vetor v com seus valores alternados.

$$v' = 1 - \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \end{bmatrix}_{1\times a} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \end{bmatrix}_{1\times a}$$

$$PV = PR1_{a\times a} * diagonal(v)_{a\times a} \; + \; PR0_{a\times a} * diagonal(v')_{a\times a}$$

$$PV = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & \frac{1}{2} & \frac{1}{2} \\ 0 & 1 & 1 & \frac{1}{2} & \frac{1}{2} \end{bmatrix}_{l\times a} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{a\times a} + \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & \frac{1}{2} & \frac{1}{2} \\ 1 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix}_{l\times a} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}_{a\times a}$$

$$PV = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 & \frac{1}{2} & \frac{1}{2} \end{bmatrix}_{l\times a}$$

O próximo passo seria então usar a matriz PV para encontrar classificação que apresenta maior valor para o vetor v. Antes disso, decidiu-se normatizar todos os valores de PV pelo função logaritmo. Entretanto, alguns valores de PV podem possuir o valor "0", pois tal valor indica que um determinado atributo para uma dada classificação nunca apareceu. Considerando que tal valor não é definido na função logaritmo, resolveu-se antes somar "1" a matriz PV e logo após calcular a matriz logaritmica dessa nova matriz.

$$PV_{l\times a} = PV_{a\times a} + 1$$

$$PV = \begin{bmatrix} 2 & 2 & 1 & 2 & 1 \\ 2 & 2 & 2 & \frac{3}{2} & \frac{3}{2} \\ 1 & 1 & 2 & \frac{3}{2} & \frac{3}{2} \end{bmatrix}_{l\times a}$$

$$PV_{l\times a} = log(PV_{l\times a})$$

$$PV = \begin{bmatrix} 0.69315 & 0.69315 & 0.00000 & 0.69315 & 0.00000 \\ 0.69315 & 0.69315 & 0.69315 & 0.40547 & 0.40547 \\ 0.00000 & 0.00000 & 0.69315 & 0.40547 & 0.40547 \end{bmatrix}_{l\times a}$$

Com a matriz PV normatizada, pode-se então obter o vetor "u". Esse vetor é representado por um vetor coluna, onde cada linha representa a soma de todos os elementos de uma linha da matriz PV, logo o vetor "u" é resultado da multiplicão da matriz PV com um vetor unitário de dimensão "a" por 1, onde "a" representa também o número de linhas da matriz PV. Originalmente, considerou-se fazer a multiplicação dos elementos da linha da matriz PV ao invés da soma, entretanto como a PV pode conter até centenas de atributos "a", o número obtido pela multiplicação pode ser muito pequeno, ocasionando problemas computacionais. Dessa forma, a adição foi preferida ao invés da multiplicação.

$$u = PV_{l\times a} * \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}_{a\times 1}$$

$$u = \begin{bmatrix} 2.07944 \\ 2.89037 \\ 1.50408 \end{bmatrix}_{l\times 1}$$

O vetor PF contendo a probabilidade de cada classificação para o vetor de entrada v é obtido pela relação entre PH com o vetor u, onde PF é o resultado da soma do vetor PH com o vetor u. Vale ressaltar que o vetor PH deve passar pelo mesmo processo do vetor PV, ou seja, somar 1 ao vetor e depois normatizar pela função logarítmica.

$$PH = 1 + PH_{l\times 1}$$

$$PH = 1 + \begin{bmatrix} \frac{1}{5} \\ \frac{2}{5} \\ \frac{2}{5} \end{bmatrix}_{l\times 1}$$

$$PH = \begin{bmatrix} \frac{6}{5} \\ \frac{7}{5} \\ \frac{7}{5} \end{bmatrix}_{l\times 1}$$

$$PH = log(PH)$$

$$PH = \begin{bmatrix} 0.18232 \\ 0.33647 \\ 0.33647 \end{bmatrix}_{l\times 1}$$

$$PF = PH_{l\times 1} + u_{l\times 1}$$

$$PF = \begin{bmatrix} 0.18232 \\ 0.33647 \\ 0.33647 \end{bmatrix}_{l\times 1} + \begin{bmatrix} 2.07944 \\ 2.89037 \\ 1.50408 \end{bmatrix}_{l\times 1}$$

$$PF = \begin{bmatrix} 2.26176 \\ 3.22684 \\ 1.84055 \end{bmatrix}_{l \times 1}$$

Por fim, a classificação que possui a maior probabilidade para o vetor de entrada v, é a classificação "1", pois a classificação "1" no vetor B está na mesma linha que a maior probabilidade no vetor PF. Logo a resposta do algoritmo para a entrada do vetor v é a classificação "1".

Relacionando a algebra linear utilizada como exemplo, à fórmula previamente explicada na Seção 2.2.2, é possível identificar cada elemento da fórmula a uma matriz ou vetor do exemplo.

$Classificador = max(p(C_y) * \prod_{i=1}^{N} p(x_i|C_y))$

Onde:

- C**:** Conjunto das classes possíveis representado pelo vetor B;
- n**:** Número total de itens, representado pela dimensão "a", que se trata do número de atributos na matriz D, ou o número de atributos no vetor "v";
- $p(C)$**:** Probabilidade de cada classe no conjunto de classes possíveis, representado pelo vetor PH;
- $p(x_i|C_y)$**:** Cálculo da probabilidade bayesiana em si, assumindo que as variáveis são independentes, representado pela matriz PV.

APÊNDICE E – Termo de consentimento livre e esclarecido - Primeiro Experimento

O (a) Senhor(a) está sendo convidado(a) a participar da pesquisa *Estratégia de contexto temporal em recomendação de pacotes GNU/Linux*

Este pesquisa tem como objetivo coletar os seguintes dados do usuário:

- Versão do sistema operacional do usuário;
- Versão do kernel do usuário;
- Versão do processador do usuário;
- Lista de todos os pacotes instalados na máquina do usuário;
- Lista de todos os pacotes manualmente instalados;
- Tempo de acesso e modificação de todos os pacotes manualmente instalados;
- Caminho de todos os pacotes manualmente instalados;
- Submissão do usuário para o sistema popularity-contest;
- Data e hora da coleta

O(a) senhor(a) receberá todos os esclarecimentos necessários antes e no decorrer da pesquisa e lhe asseguramos que seu nome não aparecerá, sendo mantido o mais rigoroso sigilo através da omissão total de quaisquer informações que permitam identificá-lo(a)

A sua participação será através da instação da aplicação AppRecommender na sua própria máquina, juntamente com a execução de um script responsável pela coleta dos dados citados como objetivo da pesquisa. Sua participação é voluntária, isto é, não há pagamento por sua colaboração.

Os resultados da pesquisa serão divulgados na primeira parte do trabalho de conclusão de curso dos alunos Lucas Albuquerque Medeiros de Moura e Luciano Prestes Calvancati.

Se o(a) Senhor(a) tiver qualquer dúvida em relação à pesquisa, por favor mande um e-mail para: lucas.moura128@gmail.com ou lucianopcbr@gmail.com.

Este documento foi elaborado em duas vias, uma ficará com o pesquisador responsável e a outra com o sujeito da pesquisa.

Ao concordar com este termo, o senhor(a) cederá os dados coletados sem recompensa financeira e para uso em pesquisa científica sem fins lucrativos apenas. Esses dados não serão repassados para terceiros.

_____

Nome/Assinatura

_____

Pesquisador Responsável

Brasília, _____ de _____ de _____

APÊNDICE F – Termo de consentimento livre e esclarecido - Segundo Experimento

O (a) Senhor(a) está sendo convidado(a) a participar da pesquisa *Estratégia de contexto temporal em recomendação de pacotes GNU/Linux*

Este pesquisa tem como objetivo coletar os seguintes dados do usuário:

- Versão do sistema operacional do usuário;
- Versão do kernel do usuário;
- Versão do processador do usuário;
- Lista de todos os pacotes instalados na máquina do usuário;
- Lista de todos os pacotes manualmente instalados;
- Tempo de acesso e modificação de todos os pacotes manualmente instalados;
- Submissão do usuário para o sistema popularity-contest;
- Data e hora da coleta

O(a) senhor(a) receberá todos os esclarecimentos necessários antes e no decorrer da pesquisa e lhe asseguramos que seu nome não aparecerá, sendo mantido o mais rigoroso sigilo através da omissão total de quaisquer informações que permitam identificá-lo(a)

Nome/Assinatura

Pesquisador Responsável

A sua participação será através da instação da aplicação AppRecommender na sua própria máquina, juntamente com a execução de um script responsável pela coleta dos dados citados como objetivo da pesquisa. Além disso, também será coletada a sua avaliação para um determinado números de pacotes gerado pela ferramenta AppRecommender. Sua participação é voluntária, isto é, não há pagamento por sua colaboração.

Os resultados da pesquisa serão divulgados na segunda parte do trabalho de conclusão de curso dos alunos Lucas Albuquerque Medeiros de Moura e Luciano Prestes Calvancati.

Se o(a) Senhor(a) tiver qualquer dúvida em relação à pesquisa, por favor mande um e-mail para: lucas.moura128@gmail.com ou lucianopcbr@gmail.com.

Este documento foi elaborado em duas vias, uma ficará com o pesquisador responsável e a outra com o sujeito da pesquisa.

Ao concordar com este termo, o senhor(a) cederá os dados coletados sem recompensa financeira e para uso em pesquisa científica sem fins lucrativos apenas. Esses dados não serão repassados para terceiros.

_____

Nome/Assinatura

_____

Pesquisador Responsável

Brasília, _____ de _____ de _____