UNIVERSIDADE
FEDERAL RURAL
DE PERNAMBUCO

Ariana Lima Guimarães

Técnicas de Comitês para a Estimação de Esforço na Correção de *Software*

Recife

2019

Ariana Lima Guimarães

Técnicas de Comitês para a Estimação de Esforço na Correção de *Software*

Monografia apresentada ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharela em Sistemas de Informação.

Universidade Federal Rural de Pernambuco – UFRPE

Departamento de Estatística e Informática

Curso de Bacharelado em Sistemas de Informação

Orientador: Prof. Dr. Rodrigo G. F. Soares

Recife

2019

Dados Internacionais de Catalogação na Publicação
Universidade Federal Rural de Pernambuco
Sistema Integrado de Bibliotecas
Gerada automaticamente, mediante os dados fornecidos pelo(a) autor(a)

G963t Guimarães, Ariana Lima
Técnicas de comitês para a estimação de esforço na correção de software / Ariana Lima Guimarães. - 2019.
56 f. : il.

Orientador: Rodrigo Gabriel Ferreira Soares.
Inclui referências.

Trabalho de Conclusão de Curso (Graduação) - Universidade Federal Rural de Pernambuco, Bacharelado em Sistemas de Informação, Recife, 2019.

1. Classificação textual. 2. Comitê de Classificadores. 3. Engenharia de Software. 4. Estimativa de esforço. 5. Extração de características. I. Soares, Rodrigo Gabriel Ferreira, orient. II. Título

CDD 004

ARIANA LIMA GUIMARÃES

TÉCNICAS DE COMITÊS PARA A ESTIMAÇÃO DE ESFORÇO NA CORREÇÃO DE SOFTWARE

Monografia apresentada ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharela em Sistemas de Informação.

Aprovada em: 10 de Dezembro de 2019.

BANCA EXAMINADORA

Rodrigo G. F. Soares (Orientador)
Departamento de Estatística e Informática
Universidade Federal Rural de Pernambuco

Cleviton Monteiro
Departamento de Estatística e Informática
Universidade Federal Rural de Pernambuco

Gilberto Cysneiros
Departamento de Estatística e Informática
Universidade Federal Rural de Pernambuco

À Gorete Lima e ao Zé Padeiro

Agradecimentos

Agradeço, primeiramente, à minha mãe e ao meu avô pelos sacrifícios que fizeram pelos meus estudos e pelos meus sonhos ao longo da minha vida. E à minha família, independente de laços sanguíneos.

Agradeço ao Prof. Dr. Rodrigo, meu orientador, que acreditou em mim e sempre esteve disposto a me ajudar, até mesmo à distância. Agradeço pelos ensinamentos desde as cadeiras no início do curso até este trabalho final, pela disponibilidade e pelo empenho.

Agradeço à UFRPE, pela estrutura na qual pude estudar e passar boa parte dos meus dias ao longo de 5 anos. Aos professores de BSI, em especial aos que deram oportunidades importantes para minha vida acadêmica e profissional, e aos que passaram diversas tardes - inclusive fora do horário de aulas - ajudando a mim e a outros estudantes a desenvolver os trabalhos e a corrigi-los.

Agradeço aos amigos que fiz durante a graduação. Aos que me deram maior apoio inicial ao ingressar no curso, quando eu não tinha nenhum conhecimento sobre programação. Aos que dividiram comigo a responsabilidade dos projetos durante a maior parte do curso, com os quais passei diversas noites em claro para conseguirmos finalizá-los. Aos que compreendiam e me auxiliavam nos meus horários malucos, para que eu também pudesse manter meu dia-a-dia como bailarina. Aos mais veteranos, que nos salvaram em diversos momentos de apuros. Àqueles que nos davam uma luz quando nos achávamos incapazes e queríamos desistir.

Vini, Bern, Pedro, Loki, Romero, Huguitu, Maia, Bel, Calouro, Chico, Coisinha, Mago, Lipe, Victor, Bruna, Uninho Furioso, …

Agradeço aos meus colegas de trabalho do CESAR, com os quais pude vivenciar muitas coisas e que também me auxiliaram para que eu pudesse conciliar meu tempo para o desenvolvimento deste trabalho. Aos que me ~~aperrearam~~, aconselharam, orientaram e mentoraram, tanto academicamente quanto profissionalmente.

Aos meus amigos em geral, meus amigos de infância e do Ensino Médio, os quais também tornaram esses 5 anos mais leves e alegres. Não somente aos que puderam estar mais presentes e disponíveis durante essa longa fase, mas também aos que estiveram mais distantes fisicamente graças ao imprevisível fluxo da vida.

A felicidade só é boa quando compartilhada. Então, a todos vocês, e a quem não foi citado aqui, mas também teve alguma contribuição neste trabalho e na minha vida acadêmica; meu muito obrigada!

“Rise, and rise again. Until lambs become lions.”
(Robin Hood)

Resumo

O planejamento bem definido de um projeto de *software*, desde os estágios iniciais, é imprescindível para o sucesso do desenvolvimento, seja ele referente à criação ou à manutenção do produto. Em anuência ao ciclo de vida de *software*, a manutenção é realizada de forma contínua após o produto ter sido construído e entregue, em paralelo à execução de testes por engenheiros e/ou usuários. Nessa etapa, surgem primariamente os documentos de Histórias de Usuário e Relatórios de Problemas, que descrevem, em linguagem natural, especificações de negócio, cenários de erros encontrados, correções esperadas e melhorias para o sistema. Esses documentos visam, dentre outras coisas, o mapeamento das atividades a serem realizadas durante o projeto. Por conseguinte, em consonância com os recursos disponíveis – humanos, financeiros e temporais -, torna-se possível estimar o esforço necessário no desenvolvimento das atividades e gerar informações essenciais a um planejamento eficaz e eficiente. Como esses documentos são escritos em textos naturais, surge a oportunidade de utilizar o Processamento de Linguagem Natural e o Aprendizado de Máquina (AM) para predição automatizada do esforço de *software*. Na prática, no dia-a-dia das fábricas de *software*, é comum a utilização da opinião de especialistas e da equipe do projeto para julgar o esforço requisitado por uma atividade durante sessões de *Planning Poker*. Nessa técnica, normalmente o esforço é medido em Pontos de História que seguem a sequência *Fibonacci*. Porém, esse modo de planejamento requer o escalonamento de muitos recursos para sua execução. A aplicação do AM acarreta em um sistema, após a fase de treinamento, capaz de apreender a experiência da equipe e replicá-la de forma rápida e automática para estimar o esforço das atividades. Dessa forma, este trabalho atinge a área de AM, propondo uma abordagem de Comitê de PV-DM na extração de características de Relatórios de Problemas para estimar Pontos de História, os indicadores de esforço. Comparada a outras duas abordagens de BoW e PV-DM tradicional, a técnica proposta apresentou bons resultados, com *f-measure* de cerca de 80% em um classificador de SVM com aprendizado supervisionado. Os resultados dos experimentos inspiram um ponto de partida no aprofundamento do estudo da abordagem de Comitê de PV-DM e no seu aprimoramento.

Palavras-chave: Classificação Textual, Comitê de Classificadores, Engenharia de Software, Estimativa de Esforço, Extração de Características.

Abstract

A well-defined planning of a software project, since the early stages, is indispensable to its success, whether the development refers to product's creation or maintenance. Accordingly to the software life cycle, maintenance is continuously executed after the product's building and delivery, in parallel to the tests execution by engineers and/or users. In this stage, User Stories and Issue Reports are the first documents to be presented. These documents describe, in natural language, business requirements, error scenarios found, expected corrections and enhancements for the system. Its objectives are, among other things, ranking the activities needed to be accomplish during the project. Therefore, in line with the available resources – human, financial and temporal -, it is possible to estimate the effort that will be necessary in the activities development and generate essential information for an effective and efficient planning. As these documents are written in natural texts, it raises the opportunity to use Natural Language Processing and Machine Learning (ML) to predict software effort. In practice, in the daily life of software factories, it is common to use experts' and project staff's opinion to judge the effort required by an activity during Planning Poker sessions. Usually, in this technique, the effort is measured in Story Points, which follow *Fibonacci* sequence. But this planning model requires the scaling of more resources to be executed. The application of ML causes in a system, after the learning phase, the ability to seize the team experience and replicate it quickly and automatically to estimate the activities effort. Thus, this work covers the ML field, proposing a PV-DM Ensemble approach to extract features of Issue Reports to estimate Story Points, the effort indicator. Compared to the two other approaches of BoW and simple PV-DM, the proposed technique has presented good results, about 80% of f-measure, in a supervised learning SVM classifier. The experiments results proved to be a starting point for further study of PV-DM Ensemble approach and its improvement.

Keywords: Effort Estimation, Ensembles, Feature Extraction, Software Engineering, Text Classification.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AM	Aprendizado de Máquina
APSTUD	Aptana Studio
BoW	*Bag of Words*
COCOMO	*Constructive Cost Model*
IA	Inteligência Artificial
IDE	*Integrated Development Environment* (Ambiente Integrado de Desenvolvimento)
IEE	*Institute of Electrical and Electronics Engineers*
LF	Linguagem Formal
LN	Linguagem Natural
ML	*Machine Learning*
MLP	*Multilayer Perceptron*
PLN	Processamento de Linguagem Natural
PP	*Planning Poker* (*Poker de Planejamento)*
PV-DM	*Paragraph Vector - Distributed Memory*
RBF	*Radial Basis Function* (Função de Base Radial)
RN	Redes Neurais
SEER-SEM	*SEER for Software*
SLIM	*Software Life-cycle Model*
SVM	*Support Vector Machine* (Máquina de Vetores Suporte)
TF-IDF	*Term-Frequency Inverse Document-Frequency*
url	*Uniform Resource Locator* (Localizador Uniforme de Recursos)

Sumário

1 Introdução

Processo, do latim *processus*: “Sequência contínua de fatos ou fenômenos que […] se reproduzem com certa regularidade […]” (WEISZFLOG, 2015), normalmente objetivando o alcance de uma meta. Para garantir que organizações - a exemplo de fábricas de *software* - executem seus projetos de sistemas de forma eficaz e eficiente, (PROJECT MANAGEMENT INSTITUTE, 2017) evidencia a importância da utilização de processos, pois eles auxiliam no cumprimento dos requisitos, dos prazos e na garantia de qualidade do produto. Assim, as organizações tornam-se mais propensas a satisfazer seus clientes, aumentar seus lucros, e/ou evitar prejuízos.

O desenvolvimento de *software* de qualidade exige, então, a definição e a execução de processos. (PRESSMAN; MAXIM, 2016) ressaltam que, no contexto da Engenharia de *Software*, um processo não é uma rígida prescrição de como desenvolver sistemas, mas sim uma metodologia flexível, capaz de se adaptar a diferentes necessidades. Assim, diferentes modelos de processos surgiram e evoluíram com o passar do tempo: Metodologias Clássicas (como Cascata, Espiral, etc.) e Metodologias Ágeis (*Scrum*, Programação Extrema, etc.) (KOSCIANSKI, 2007). Apesar desses processos terem diferentes fluxos, (PRESSMAN; MAXIM, 2016) afirmam que todos eles possuem um mesmo conjunto de atividades genéricas, constituído por comunicação, planejamento, modelagem, construção e implantação.

(BOURQUE; FAIRLEY, 2014) indicam que o primeiro passo no planejamento de projetos de *software* deve ser a seleção de um modelo de processo e possivelmente sua adaptação. Pode-se dizer, portanto, que o planejamento do *software* é essencial para que seja possível estimar recursos necessários nos campos de escopo, equipe, tempo e custo. Segundo (SOARES, 2018), se os recursos para um projeto são subestimados, pode haver aumento nos custos durante o desenvolvimento e atrasos na entrega. E no cenário oposto em que os recursos são superestimados, a organização pode perder vantagem competitiva e sua posição no mercado.

Durante a elaboração do planejamento, antes de estimar o tempo de desenvolvimento de uma atividade, é preciso estimar o tamanho da atividade. (MARTIN, 2005) cita o uso de Pontos de História (*Story Points*) como uma unidade de medida relativa para definir o tamanho de Histórias de Usuário, funcionalidades do sistema, ou outras partes do projeto.

“Uma estimativa eficaz é muito importante para o sucesso do desenvolvimento do *software*, evitando custos adicionais e garantindo a entrega no prazo” (JúNIOR, 2019). Então, em um cenário ideal, os recursos e esforços despendidos para um *soft-*

ware devem ser previstos antes do início do desenvolvimento propriamente dito. Na correção de *software*, uma atividade que também é considerada como sendo desenvolvimento de *software* (PORRU et al., 2016), Relatórios de Erros (*Issue Reports*) são os primeiros documentos a serem disponibilizados. Esses documentos, escritos em linguagem natural, descrevem os erros encontrados na execução do programa e, muitas vezes, o cenário que era o real esperado. Cada cenário descrito corresponde a uma atividade a ser desenvolvida para correção do problema, que requer planejamento e métricas para estimativas igualmente à construção do produto em si.

Nesse sentido, (AGGARWAL; ZHAI, 2012) apresentam a importância e a grande gama de aplicações na categorização de documentos em linguagem natural. Atividades como organização de documentos, diferentes tipos de filtros e mineração de opinião têm sido beneficiadas com tais estudos. Esses autores também destacam que algoritmos de Aprendizado de Máquina (AM), como Redes Neurais (RN) e Máquinas de Vetores Suporte (*Support Vector Machine - SVM*), estão dentre as principais ferramentas utilizadas para a categorização. (PORRU et al., 2016), por exemplo, expõem que é factível utilizar um classificador de AM para estimar os Pontos de Histórias para um problema de *software* a partir de recursos como resumo e descrição desse.

> Atualmente, os clientes estão cada vez mais exigentes e procuram fábricas que possuam um rigoroso controle de qualidade. […] A exigência dos clientes faz as organizações selarem contratos que medem o nível da qualidade de seus serviços. Esses contratos prevêem como sanções pesadas multas.
>
> O cumprimento de prazos e custos é uma das exigências previstas nesse tipo de contrato. A base para que se cumpram prazos e custos é a estimativa de esforço de desenvolvimento e, após a entrega do produto, o esforço para a manutenção do software […] (JUNIOR, 2010).

Assim sendo, considerando a importância da predição de esforço para desenvolvimento e/ou correção de *software* de forma rápida e precisa e o atual cenário da classificação de linguagem natural utilizada como base para tais experimentos, este trabalho visa fazer uma comparação de extração de atributos e características para predição de esforço. Com base em Relatórios de Problemas de *softwares* já existentes, será proposta uma nova metodologia para extração de características, a qual será comparada com outras duas abordagens:

- **Abordagem 1:** *Bag of Words* (BoW)
- **Abordagem 2:** *Paragraph Vector - Distributed Memory* (PV-DM)
- **Abordagem 3 (metodologia proposta):** Comitê de Classificadores PV-DM

Pode-se afirmar, portanto, que este trabalho tem como objetivo auxiliar equipes de projetos de *software* a estimarem o esforço a ser despendido para suas atividades de desenvolvimento. A partir da comparação das abordagens de extração de características citadas anteriormente, levanta-se a hipótese de que o Aprendizado de Máquina é capaz de replicar o conhecimento humano de estimar a complexidade de atividades a partir de suas descrições. Logo, a automatização desse processo permitirá a execução de um planejamento adequado e eficiente.

O próximo capítulo irá detalhar as problemáticas na estimativa de esforço de *software*, como é feito atualmente nas fábricas de *software* e como o Aprendizado de Máquina pode trazer benefícios para as equipes de desenvolvimento. No capítulo 3 são exibidos os trabalhos relacionados ao tema e algumas das variadas técnicas para estimativa de esforço. O capítulo 4 apresenta o referencial teórico que embasa este trabalho: técnicas de Processamento de Linguagem Natural e técnicas de Inteligência Artificial que permitam a classificação textual nos experimentos. O capítulo 5 apresenta a proposta da abordagem de Comitê de PV-DM e, além dela, descreve as abordagens de BoW e PV-DM simples para extração características textuais. No capítulo 6 são trazidos detalhes dos experimentos, características específicas da base de dados, os resultados obtidos e a análise deles. Por fim, o capítulo 7 conclui o trabalho retomando a problemática em discussão, debatendo brevemente os resultados obtidos em cada abordagem de extração de características e levantando possíveis trabalhos futuros.

2 Estimativa de Esforço

De acordo com (SOMMERVILLE, 2011), a atividade de planejamento de projetos deve gerar estimativas, estrutura de divisão do trabalho, definição de equipe e demais recursos, alocação de pessoas por atividade, cronograma e orçamento. Essas informações, por sua vez, são possíveis de serem geradas - direta ou indiretamente - a partir da estimativa de esforço de *software*.

No capítulo anterior, já foi evidenciada a importância do planejamento de projetos de *software*. Sua ausência pode desencadear problemas graves ao projeto e até à organização como um todo, em efeito cascata. Sem planejamento não há prazos nem estimativa de funcionalidades a serem implementadas, sem estimativas não é possível dimensionar a quantidade de atividades a serem executadas nem o esforço necessário, e sem orçamento não se tem noção dos custos relacionados ao projeto e não se sabe se os recursos disponíveis serão suficientes para sua execução.

Um processo de *software* é resultado do planejamento associado a um modelo de processo. É de extrema importância o alinhamento dessas atividades, pois o modelo de processo irá ditar quais atividades devem ser realizadas e a ordem em que devem ser executadas para atender às dependências. Ou seja, o modelo influencia diretamente no planejamento e nas estimativas. Por exemplo, no Modelo Cascata (PRESSMAN; MAXIM, 2016), é necessária a especificação de uma série de documentos e diagramas (requisitos, arquitetura, *design*, etc.) antes do início da implementação. O planejamento deve indicar quem faz estas atividades, quanto tempo é necessário para elas, quando elas serão realizadas e quanto custará cada uma delas. Já nas Metodologias Ágeis (PRESSMAN; MAXIM, 2016), mais comuns hoje em dia, o processo é definido em ciclos, com a liberação de versões da aplicação de forma mais frequente e com menos burocracia, prezando a flexibilidade do projeto. Com isso, verifica-se uma crescente tendência de redução na elaboração de documentos e/ou de elaboração de documentos mais sucintos.

Na manutenção de *software*, especificamente, é comum a criação de documentos de Histórias de Usuário e Relatórios de Problemas desde o início dos projetos. As histórias definem o escopo do projeto com base nos requisitos de negócio, explicitando quem são os usuários da aplicação, as ações que eles farão e o intuito das ações. A partir de documentos desse tipo, equipes especializadas em testar a aplicação são capazes de entender o universo no qual a aplicação se encontra, os requisitos desejados pelo cliente e os possíveis cenários de erro. Tendo sido os cenários elencados, baterias de testes são executadas sobre o sistema e Relatórios de Problemas são gerados.

Em sequência, tais relatórios, além de descreverem os cenários de erros encontrados, expõem as correções esperadas e melhorias para o sistema (incrementação de funcionalidades existentes ou implementação de novas funcionalidades) em linguagem natural. Segundo (PATTON, 2006), o custo de encontrar defeitos no sistema e removê-los cresce de forma exponencial. Portanto, quanto antes houver o mapeamento das atividades, o planejamento para execução - que envolve a estimativa de esforço - e o desenvolvimento da atividade (a correção), reduzem-se os custos para o projeto.

(JúNIOR, 2019) reconhece que a estimativa de esforço é responsabilidade da equipe de desenvolvimento, já que ela é detentora de conhecimento sobre a cultura da organização para a qual trabalha. A experiência do time com o cliente, com o produto, e com os colegas de trabalho é o que permite estimar o esforço a ser despendido para as tarefas. O *Planning Poker* (PP), segundo (RUBIN, 2012), é uma técnica que visa mensurar atividades a serem realizadas para um produto que se popularizou por volta do ano 2006.

O PP é uma técnica baseada em consenso, na qual toda a equipe do projeto ou pessoas designadas debatem e expõem suposições para adquirir conhecimento e estimar as atividades a serem realizadas. Quase como um jogo, para cada atividade prevista, esclarecem-se os detalhes para alinhamento do entendimento dessa, e então cada participante exibe um cartão com um valor para a atividade. Caso haja um consenso na escolha do valor, ele se torna a estimativa prevista para a atividade. Caso contrário, retoma-se o debate sobre a atividade e faz-se uma nova eleição, até que haja um consenso. Os valores seguem o início da sequência de *Fibonacci* (equação 2.1), definida recursivamente na fórmula 2.2, com os valores iniciais iguais a 1.

$$1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots \tag{2.1}$$

$$F_n = F_{n-1} + F_{n-2} \tag{2.2}$$

A utilização da sequência *Fibonacci*, ao invés do conjunto $\mathbb{Z}_+$, deve-se ao fato de que os “saltos” que existem entre os termos da sequência vão medir a incerteza de uma estimativa, além de facilitar a escolha quando se há dúvida entre números próximos na sequência. Por exemplo, caso a equipe divirja as opiniões sobre uma atividade entre os Pontos 8 e 13, é mais fácil chegar em um consenso entre esses dois números do que entre 8 e 9 que estão muito mais próximos e seus valores são mais difíceis de serem diferenciados.

Os valores definidos para cada atividade correspondem, portanto, aos seus Pontos de História, que denotam a dificuldade esperada para execução de uma determinada tarefa pelo time de desenvolvimento (SOARES, 2018). Quanto mais baixo o valor, mais simples a atividade, e quanto mais alto, mais complexa.

Observa-se, assim, que o principal objetivo direto da estimativa é determinar o tempo de desenvolvimento necessário para uma atividade. Esse tempo depende do tamanho do *software* e da atividade em si, da sua complexidade e da produtividade dos desenvolvedores. Com tantas dependências bastante subjuntivas, é de se esperar que as estimativas apresentem certo grau de incerteza (PROJECT MANAGEMENT INSTITUTE, 2017). Até porque, sempre podem ocorrer imprevistos que aumentem ou reduzam o esforço estimado para uma tarefa. (GALORATH; EVANS, 2006) alertam, ainda, que como as estimativas são baseadas em conhecimento e suposições incompletas e imperfeitas sobre o futuro, muitas vezes o consenso da equipe tende a subestimar o esforço da atividade devido a omissões de importantes funções do produto e requisitos do projeto.

A estimação manual também apresenta outra grande problemática: a dependência do conhecimento e experiência restritas a funcionários específicos. Caso esses funcionários não estejam mais disponíveis para o projeto ou para a empresa, a capacidade de análise e estimação de esforço é afetada.

Não obstante, muitas empresas visam apenas a produtividade dos funcionários, não dando a devida importância às fases de planejamento e aos riscos que existem na ausência deles. Como em um cenário ideal o planejamento não deve deixar de ser feito, Metodologias Ágeis sugerem processos mais enxutos nessa etapa, para que haja mais tempo para produção. A automatização do processo de esforço de *software* surge para sanar essa questão, pois o planejamento não deixa de ser feito, mas acontece muito mais rápido, atendendo aos desejos organizacionais e mantendo um ambiente de trabalho saudável e organizado para os funcionários.

3 Trabalhos Relacionados

O conceito de estimar esforço de *software*, conforme (ZIA; RASHID; ZAMAN, 2011) - citado por (ZIAUDDIN; ZIA, 2012) -, existe desde o início dos anos 50, quando os desenvolvedores e pesquisadores iniciaram suas tentativas de criar métodos para estimar custos e cronogramas para o desenvolvimento de projetos *software*. Atualmente, (SAINI, 2014) demonstra existir uma variedade de técnicas para estimar esforço de *software*:

- Julgamento de Especialistas
- Técnicas baseadas em Analogia
- Técnicas Algorítmicas
- Aprendizado de Máquina

(JØRGENSEN, 2005) apresenta sete diretrizes para produção realista de estimativas de esforço de *software* baseado no Julgamento de Especialistas, pois é a metodologia mais comum nos dias de hoje. No entanto, essa metodologia tende a ser um processo que envolve um certo grau da inconsciência humana. (GRIMSTAD; JØRGENSEN, 2007) alertam que uma mesma informação apresentada a um mesmo indivíduo, em diferentes ocasiões, pode levar a diferentes resultados. Portanto, estimativas de *software* baseadas em especialistas tendem a apresentar alto grau de inconsistência. Nos experimentos realizados no artigo, os autores reportaram cerca de 71% de diferença nas estimativas de uma mesma atividade. Tal inconsistência representa riscos para o desenvolvimento e para a qualidade do projeto.

Projetos estimados com base em Analogia utilizam-se das métricas estimadas e praticadas de projetos anteriores. A dispensabilidade de especialistas torna o seu uso mais barato, porém requer uma disponibilidade de histórico de informações de projetos semelhantes, tornando-se difícil ou impossível de ser utilizada para projetos pioneiros e inovadores. Além do baixo custo, (HILL; THOMAS; ALLEN, 2000) citam vantagens como simplicidade relativa e precisão razoável. Já (YEATES, 1986) enfatiza a dificuldade não só de haver histórico disponível de outros projetos, mas que dificilmente eles irão corresponder exatamente aos requisitos e funcionalidades esperadas dos novos projetos, de forma que a estimativa terá de ser ajustada. Como provavelmente especialistas é que farão esses ajustes, o método acaba perdendo uma de suas vantagens já que sua utilização sozinha pode não ser recomendada. Esse método parece, então,

ser melhor utilizado como uma ferramenta de auxílio para o Julgamento de Especialistas do que uma metodologia independente.

Modelos Algorítmicos são projetados para prover uma equação matemática para estimar esforço de *software*. Segundo (OSMANBEGOVIĆ; SULJIĆ; AGIĆ, 2017), essas equações são fundamentadas em um histórico de dados de projetos anteriores e usam diferentes medidas como parâmetros de entrada, baseadas em informações como quantidade de linhas de código, pontos de função, etc. COCOMO (BOEHM, 1981), SLIM (PUTNAM; MYERS, 1992) e SEER-SEM (JENSEN, 1983) são alguns dos modelos presentes na literatura, os quais se diferenciam basicamente pela forma de seleção das funções e dos fatores de custo. Esses fatores podem ser difíceis de mensurar, pois muitas vezes são relativos e subjetivos. Podem variar desde características do produto até especificações de *hardware* e/ou do nível de técnico dos desenvolvedores. E, não obstante, como a metodologia depende de um histórico de dados, as informações dos fatores selecionados precisam ter sido coletadas durante a execução dos projetos anteriores contidos no banco de dados. (OSMANBEGOVIĆ; SULJIĆ; AGIĆ, 2017) citam dentre as vantagens do método a objetividade, a replicabilidade e a possibilidade de calibração dada as experiências anteriores; e dentre as desvantagens, a robustez que dificulta a adaptação em novos ambientes de desenvolvimento.

“Diversos estudos automatizaram a estimação de esforço no desenvolvimento de *software* para produzir predições mais eficientes e com mais acurácia” (SOARES, 2018). Algoritmos de Aprendizado de Máquina utilizam-se de padrões encontrados na base de dados para classificar as instâncias em determinadas categorias. Essas bases são, na maioria das vezes - a exemplo dos trabalhos (MINKU; YAO, 2013a; MINKU; YAO, 2013b; MINKU; YAO, 2017; MINKU; HOU, 2017) -, compostas de indicadores numéricos como custo e quantidade de linhas de código. No entanto, essas características dificilmente estão disponíveis nos estágios iniciais do desenvolvimento de *software* ágil (LEE, 2012). Sendo assim, estas formas de estimativas dificultam e/ou impedem o planejamento adequado do projeto.

Em contraposição, alguns documentos textuais - como Histórias de Usuário e Relatórios de Erros - estão disponíveis ainda no início dos projetos. Esses documentos podem trazer informações relevantes para a estimativa de esforço, sendo, portanto, uma contribuição relevante para planejamentos mais acurados. (SOARES, 2018; UEHARA, 2019; JúNIOR, 2019; PORRU et al., 2016) atacam o problema de predição de esforço utilizando como base documentos escritos em linguagem natural.

A classificação baseada em textos é bastante comum na área de AM e exige a execução de uma série de pré-processamentos no texto “cru” antes da classificação em si. (JúNIOR, 2019) afirma que os métodos do modelo baseado no Aprendizado de Máquina, focados nos efeitos das técnicas de pré-processamento de dados no con-

texto da estimativa do esforço do *software*, ganharam popularidade nos últimos anos. Isso foi possível porque esses modelos são capazes de generalizar estimativas em problemas em que há alto grau de incerteza (BISHOP, 2006).

Apesar de abordar a classificação textual, em (PORRU et al., 2016) não foi analisado o impacto de diferentes métodos de extração de características. (UEHARA, 2019) também analisa classificadores, baseados nos textos das *descrições* de problemas em Relatórios de Erros. Já em (SOARES, 2018), utilizando-se do *título* das ocorrências de seis diferentes conjuntos de Relatórios de Erros, foi feita uma análise para três métodos de extração de características: TF-IDF (MANNING; RAGHAVAN; SCHüTZE, 2009), PV-DM (LE; MIKOLOV, 2014) e *Autoencoders* (VINCENT et al., 2008). Igualmente, (JúNIOR, 2019) faz uma comparação dos mesmos métodos de extração de características textuais no *título* dos problemas existentes em quatro diferentes conjuntos de relatórios.

O presente trabalho utiliza uma das bases de dados utilizada em (SOARES, 2018; JúNIOR, 2019) e um dos métodos de extração de características também utilizados por esses autores, o PV-DM. Porém, aqui se propõe a utilização da *descrição* dos problemas contidos na base de dados. O foco será a forma como as características serão extraídas. E, por isso, o PV-DM será comparado com o BoW e com um Comitê de PV-DMs. Espera-se que o estudo possa auxiliar equipes de desenvolvimento de *software* a estimar o esforço para correções de problemas desde os estágios iniciais do projeto, como durante o *Scrum Poker* (GRENNING, 2002). Visando, assim, um planejamento mais eficaz, redução de riscos e de custos.

4 Técnicas para Classificação Textual

Conforme supracitado, as falhas - habitualmente referidas como *bugs* - encontradas nos projetos de *software* são reportadas textualmente nos Relatórios de Erros. Esses relatórios são formados por diversos documentos, como ilustrado na Figura 1, que não costumam ter padrões de escrita definidos. A figura exibe três documentos relacionados a um mesmo projeto nos quais os exemplos 1a e 1b são mais descritivos e 1c é bem objetivo, sem muitos detalhes. Também é oportuno observar que 1a e 1c são textos "corridos" enquanto 1b possui uma estrutura de passos para reprodução do cenário, resultados obtidos e resultados esperados. Não obstante, a linguagem presente nos textos também é diversa, podendo ser de cunho coloquial ou seguir a norma-padrão do idioma utilizado.

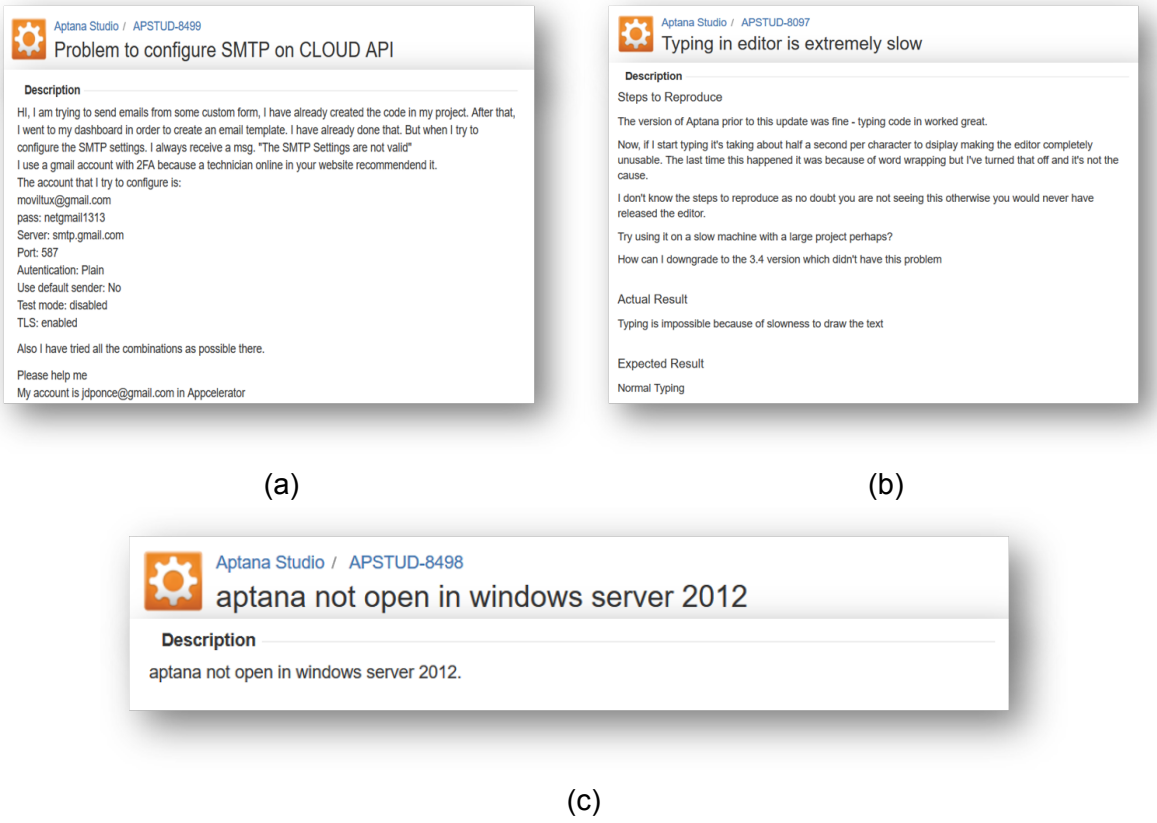


Figura 1 – Exemplos de problemas reportados em ferramentas rastreadoras *bugs*
Disponível em: <https://jira.appcelerator.org/issues/>
Acesso em 20 dez. 2019

(WEISZFLOG, 2015) traz diversas definições para o termo "linguagem", entre elas: "Sistema de sinais desenvolvido a partir de uma língua e que funciona como meio de comunicação [...]". A linguagem como forma de comunicação segue um universo

pré-definido de símbolos e um conjunto de regras ou protocolos, que visam garantir a transmissão da informação de forma clara e correta. Ela subdivide-se em linguagem natural (LN) e linguagem formal (LF). A primeira caracteriza-se por sinais instintivos aos seres humanos, a fala e a escrita, constantemente modificada e incrementada durante nossa evolução. Os diferentes idiomas existentes determinam os universos, que especificam seus alfabetos, suas estruturas gramaticais e seus vocabulários (conjunto de palavras existentes que compõe o idioma). A segunda, a linguagem formal, é composta de modelos matemáticos, é uma forma de representação - finita - e especificação da linguagem fundamentada na Teoria da Computação.

Algoritmos de Inteligência Artificial (IA) são amplamente utilizados para classificação de atributos numéricos. Não obstante, tais algoritmos também são capazes de classificar textos naturais, com suas regras sintáticas e semânticas. (LEITãO, 2006) cita que descrições em LN podem ser ambíguas e/ou inconsistentes, dependendo da interpretação do leitor. Portanto, para que os algoritmos - escritos em linguagem formal - consigam compreender a LN, faz-se necessário aplicar técnicas de pré-processamento sobre os textos na sua forma original (“crua”), para que se tornem mais similares à LF. A seguir, serão descritas as técnicas de pré-processamento necessárias, os algoritmos de IA que serão posteriormente utilizados para a extração de características e/ou para classificação, e métodos de validação e avaliação dos classificadores.

4.1 Técnicas para Pré-Processamento

O Processamento da Linguagem Natural (PLN) (ALLEN, 1987) é uma subárea da Inteligência Artificial, normalmente focada na interpretação ou produção de textos. A interpretação está relacionada ao desejo de converter linguagem natural em formal, enquanto na produção visa-se o caso contrário. Para introdução de textos a algoritmos de IA, é necessária a interpretação desses. A preparação do texto natural deve passar desde a remoção de ruídos até tratamentos mais complexos, como lematização. A seguir serão descritas as técnicas necessárias para o PLN.

4.1.1 Tokenização

Inicialmente, é preciso estruturar a fonte de dados. A tokenização tem como objetivo separar palavras ou sentenças em unidades, podendo ser lexical ou sentencial. A tokenização lexical separa cada palavra como um *token* no texto, por exemplo:

```
Sentença original: Apesar de o sistema móvel estar funcionando,
ele deve funcionar em múltiplas plataformas.
Tokenização lexical: ["Apesar", "de", "o", "sistema", "móvel",
```

```
"estar", "funcionando", ",", "ele", "deve", "funcionar", "em",
"múltiplas", "plataformas", "."]
```

No caso da tokenização sentencial, as sentenças é que são identificadas e marcadas:

```
Sentenças originais: Apesar de o sistema móvel estar funcionando, ele
deve funcionar em múltiplas plataformas. Esta é a segunda sentença.
Tokenização sentencial: ["Apesar de o sistema móvel estar funcionando,
ele deve funcionar em múltiplas plataformas.", "Esta é a segunda
sentença."]
```

Os próximos passos de PLN atuam nas unidades tokenizadas.

4.1.2 Remoção de Ruído

"[...] descrições em LN podem ser ambíguas e/ou inconsistentes. [...] Uma má interpretação dos artefatos de *software* pode levar a erros na fase de codificação e de execução dos testes" (LEITãO, 2006). Não suficiente, é bastante comum que os artefatos apresentem erros sintáticos, capazes de confundir os algoritmos, fazendo-os achar que os termos "sistema" e "sistema*a*", por exemplo, são duas palavras válidas e distintas. A LN também possui termos que devem ser usados para cumprimento das regras de cada idioma, mas que demonstram pouco ou nenhum significado na prática. Todos esses pontos são considerados ruídos e quanto menos houver na base de dados, melhor será o aprendizado.

4.1.2.1 Expressão Regular

Expressão regular é uma ferramenta da computação que facilita a identificação de cadeias de caracteres específicas. Podem ser utilizadas para filtrar ou validar trechos de textos. Identificar erros de digitação por meio de expressão regular seria muito custoso devido à complexidade da operação, mas a ferramenta pode ser utilizada para remoção de outros tipos de ruídos.

Relatórios de Erros normalmente possuem trechos de código e *log* de erros em suas descrições, muitas vezes com marcadores específicos de início e fim dos trechos. Os documentos também podem ter *urls*, *e-mails*, caracteres especiais (como "!", "?", "#", etc.) e outros termos que não fazem parte do alfabeto da LN. Expressões regulares são capazes de identificar esses termos, que podem ser removidos, como feito em (UEHARA, 2019), ou substituídos por outros termos padrões que possam ser interpretados para LF.

Aplicação no exemplo a seguir demonstra a remoção da vírgula e do ponto final - os caracteres especiais - presentes na sentença original tokenizada:

```
Sentença original: Apesar de o sistema móvel estar funcionando,
ele deve funcionar em múltiplas plataformas.
Remoção de Ruídos: ["Apesar", "de", "o", "sistema", "móvel",
"estar", "funcionando", "ele", "deve", "funcionar", "em",
"múltiplas", "plataformas"]
```

4.1.2.2 *Stop Words*

Também chamadas de palavras vazias, as *stop words* são, normalmente, as palavras mais comuns da língua e as palavras funcionais (preposição, conjunções e determinantes). É válido ressaltar que não existe uma lista universal de *stop words*. A regra básica para uma palavra ser considerada vazia é a incapacidade de discriminar entre as diferentes classes. Portanto, qualquer conjunto de palavras pode ser uma lista de *stop words* dependendo do propósito e do contexto.

Em alguns casos, como busca e indexação, remover todas as *stop words* pode ser um problema. Fazendo uma busca *online* pelo livro "A Hospedeira" (MEYER, 2009), os primeiros resultados são, de fato, referentes ao livro. Porém, removendo o artigo "A" (uma provável *stop word*) e fazendo a busca apenas com o termo "Hospedeira", os primeiros resultados são relacionados à Biologia, incongruente com o esperado no caso. Para objetivos de classificação, a alta ocorrência das palavras funcionais pode fazer o algoritmo dar muita importância ao termo, atrapalhando a análise. Portanto, para a proposta em questão, mostra-se adequado removê-las.

Aplicação no exemplo com a remoção de *stop words*, tendo sido considerados artigos, pronomes, preposições e advérbios:

```
Sentença original: Apesar de o sistema móvel estar funcionando,
ele deve funcionar em múltiplas plataformas.
Remoção de Stop Words: ["sistema", "móvel", "estar", "funcionando",
"deve", "funcionar", "múltiplas", "plataformas"]
```

4.1.3 Lematização

Lematizar: "Reduzir uma palavra flexionada à sua parte essencial" (WEISZFLOG, 2015). O objetivo com essa técnica é abstrair as variações de uma mesma palavra, como conjugações de verbo ou grau e gênero de substantivos, a um radical comum. Por exemplo, as palavras "tiver", "tenho", "tinha", "tem"; todas possuem o mesmo lema: "ter". Na investigação linguística, esses tipos de flexões não agregam

informação ao vocábulo, são irrelevantes. Portanto, para classificação textual é preferível utilizar o lema das palavras ao invés das palavras originais.

O processo de lematização, por ser uma forma de compressão do vocabulário, também auxilia na manutenção da dimensionalidade do problema, pois menos palavras serão consideradas como parte do vocabulário. Por conseguinte, o processo também auxilia no desempenho dos algoritmos.

Aplicação no exemplo:

```
Sentença original: Apesar de o sistema móvel estar funcionando,
ele deve funcionar em múltiplas plataformas.
Lematização: ["sistema", "móvel", "estar", "funcionar", "dever",
"funcionar", "múltipla", "plataforma"]
```

4.2 Técnicas de IA

(KAPLAN; HAENLEIN, 2019) definem a Inteligência Artificial como uma capacidade do sistema de interpretar dados externos, aprender a partir deles e utilizar esses conhecimentos para atingir objetivos específicos através de adaptação flexível. O Aprendizado de Máquina, então, busca aprimorar a capacidade do sistema de generalização a partir da inteligência.

O AM pode ser supervisionado ou não supervisionado. No primeiro, o conjunto de dados possui rótulos conhecidos, que atuam como supervisores do aprendizado. É possível rotular novas instâncias, classificá-las dentro de determinadas categorias. Já no segundo, não há rótulos, então é um trabalho mais explorativo em busca de semelhanças, agrupamentos ou associações. Este trabalho irá abordar o Aprendizado Supervisionado, no qual as instâncias possuem rótulos para análise do aprendizado.

Uma grande problemática da área é a disponibilidade de instâncias. Um aprendizado só pode ser, de fato, perfeito, se todas as instâncias do universo fossem apresentadas ao sistema para treinamento, de forma que ele obteria conhecimento total. No entanto, na prática, isso não é possível. O que acontece, na verdade, é um processo de amostragem de instâncias. Uma amostra do universo completo é selecionada e apresentada ao sistema para treinamento. Quanto mais instâncias presentes no conjunto, mais fácil é o aprendizado, pois o algoritmo terá maior conhecimento sobre a diversidade existente. Por outro lado, quanto mais atributos estiverem relacionados ao problema, mais difícil se tornará o aprendizado, pois mais relações precisam ser feitas para que um padrão seja encontrado.

Dessa maneira, o PLN tende a ser intrinsecamente mais difícil do que outros tipos de processamento, pois os seus atributos são compostos pelos vocabulários exis-

tentes no idioma dos documentos apresentados ao sistema. (NUNES, 2018), ao citar (BAGNO, 2012), notabiliza a riqueza que é uma língua humana viva. Por ser viva, a língua está em constante modificação. Isso, somado ao fato de que os idiomas são constituídos por uma enorme gama de palavras (incluindo muitos sinônimos), cria uma tendência de que os atributos sejam matrizes enormes e extremamente esparsas.

Assim, a diversidade da língua permite aos seus falantes expressarem uma mesma opinião de várias formas diferentes. Mas, para o PLN, isso gera um problema, pois existe uma grande quantidade de palavras no vocabulário, e muitas delas são usadas pouquíssimas vezes. O próximo capítulo irá aprofundar essa problemática de seleção e extração de atributos, visto que documentos textuais serão utilizados para estimação do esforço de *software*. Nas seções a seguir, serão descritos os algoritmos de IA que serão abordados nos experimentos deste trabalho.

4.2.1 Redes Neurais

As Redes Neurais são inspiradas na natureza, no cérebro que é capaz de pensar e reconhecer padrões. Criou-se, então, a partir do neurônio real (Figura 2), uma abstração matemática, exposta na Figura 3.

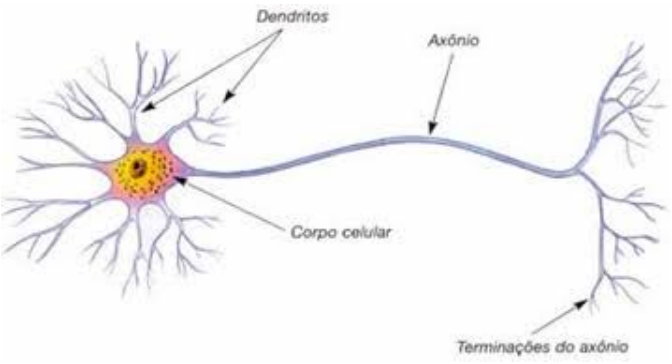


Figura 2 – Representação de um neurônio real
Fonte: (Só BIOLOGIA. VIRTUOUS TECNOLOGIA DA INFORMAçãO, 2008-2019)

$$\sum = \text{net input} = \sum_{i=0}^{n} W_i X_i \tag{4.1}$$

Quando um neurônio recebe um estímulo, se tiver sido forte o suficiente para atingir seu limiar de ativação, esse estímulo é transmitido para os neurônios adjacentes, até se dissipar. A função de net, dada pela fórmula 4.1, é o somatório do sinal proveniente de todos os outros neurônios anteriores. $f(net)$ pode ser qualquer função continuamente diferenciável, como sigmoide logística, *softmax*, tangente hiperbólica, entre outras.

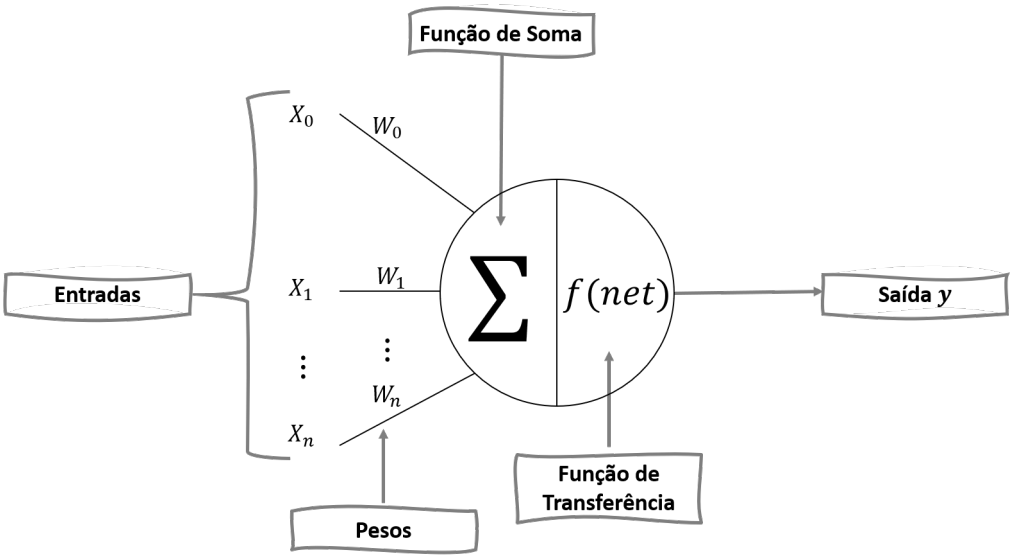


Figura 3 – Representação de um neurônio artificial

O treinamento da Rede Neural consiste em encontrar os pesos que melhor se ajustem aos dados. O conhecimento apreendido não é naturalmente compreensível aos olhos humanos, ele fica encapsulado nos pesos W, de forma que o objetivo ao ajustá-los é minimizar a função de erro E relativa à saída esperada. Quanto menor a taxa de aprendizagem, mais preciso é o algoritmo. A função 4.2 é uma função do erro em que d é o valor desejado e f é o valor obtido, porém ela só funciona numa função convexa, com um único mínimo.

$$E(W) = \frac{1}{2}(d - f)^2 \tag{4.2}$$

Para uma função com vários mínimos, é preciso utilizar o vetor gradiente. O vetor gradiente encontra a maior subida. Como o desejado é encontrar o erro, é preciso multiplicá-lo por -1 pra encontrar a maior descida. E para que o salto não seja tão grande, é preciso pegar apenas uma proporção desse vetor. Assim, pode-se calcular a função de erro a partir de 4.9, obtida pela dedução das fórmulas 4.3 a 4.8. W é o peso em um tempo t, obtido pelo somatório do produto entre os pesos W_i e as entradas da rede X_i.

$$W(t) = \sum_{i=1}^{n} W_i X_i \tag{4.3}$$

$$W(t+1) = W(t) + \Delta W(t) \tag{4.4}$$

$$W(t) = W(t-1) + \Delta W \tag{4.5}$$

$$\Delta W \sim -\frac{\delta E(W)}{\delta W} \text{ (oposto ao gradiente)} \tag{4.6}$$

$$\Delta W = \eta \frac{\delta E(W)}{\delta W} \qquad 0 < \eta < 1 \tag{4.7}$$

$$\frac{\delta E(W_i)}{\delta W_i} = \frac{\delta E(W_i)}{\delta f(net)} \cdot \frac{\delta f(net)}{\delta net} \cdot \frac{\delta net}{\delta W_i} \tag{4.8}$$

$$\frac{\delta E(W_i)}{\delta W_i} = f(net) - d \tag{4.9}$$

Na arquitetura *Multilayer Perceptron* (MLP), ilustrada na Figura 4, uma rede neural é uma sequência de camadas de neurônios. Nessa arquitetura uma camada é completamente conectada à camada adjacente, ou seja, todo neurônio tem ligação com todos os outros neurônios da camada anterior e posterior.

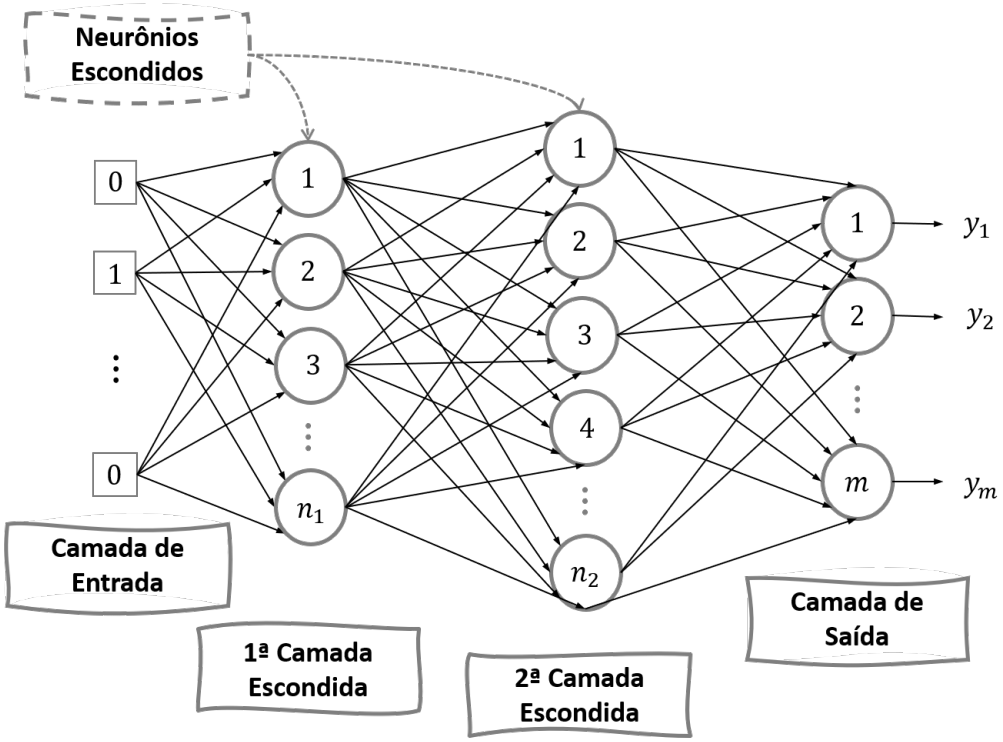


Figura 4 – Representação de uma Rede Neural

A quantidade de dimensões do problema é definida pela quantidade de X, exibidos na Figura 3. A camada de saída pode entregar um sinal ou uma probabilidade, dependendo da formulação da entrada. A quantidade de pesos de um neurônio é definida pela quantidade de neurônios da camada anterior, e a entrada de um neurônio é a saída da camada anterior.

Os parâmetros da RN são os pesos - ajustados no treinamento -, e os hiperparâmetros setados pelo projetista são: quantidade de nós na camada escondida e quantidade de camadas escondidas, taxa de aprendizagem, função de ativação de cada neurônio e taxa de momento.

No treinamento através do método de correção de erros por *Backpropagation*, o erro é propagado para trás, tal que seja calculado o δ - influência do neurônio no erro final - de cada neurônio. Assim, os pesos são atualizados e correção do erro é feita com o gradiente descendente. Se a base de dados utilizada for muito pequena, a RN não consegue aprender direito.

A Figura 5 expõe um exemplo clássico da aplicação de uma RN, na classificação de imagens de dígitos escritos à mão. Nesse caso, as instâncias do conjunto de dados, as imagens, são compostas de pixeis que representam a tonalidade de cor presente. Como a imagem possui resolução de $28 * 28$ pixeis, cada instância é uma matriz numérica de 784 pixeis que serão processados pelas camadas escondidas e a probabilidade de a matriz representar cada dígito (0 ao 9) é calculada na camada de saída. O neurônio com maior valor de probabilidade na camada de saída representa a classe predita pela Rede, 8 no caso.

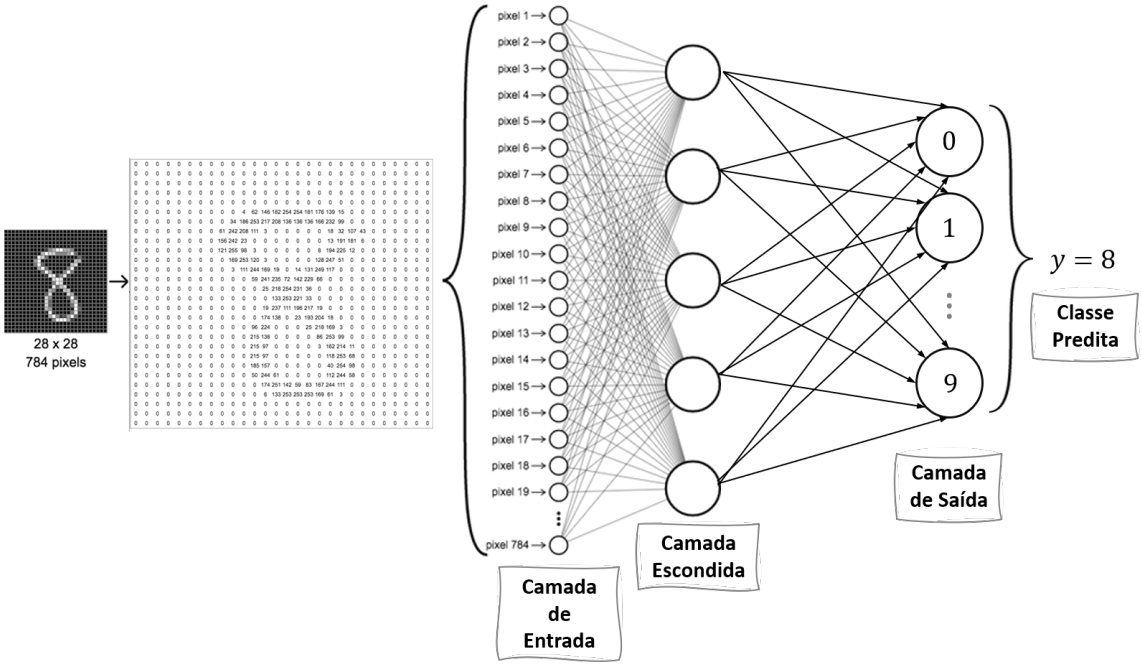


Figura 5 – Exemplo de uma RN para classificação de dígitos manuscritos
Fonte: Adaptado de (MAURYA, 2018)
Original disponível em:
<https://miro.medium.com/max/1200/1*av47vApmzuM0AN21VaIcSA.png>
Acesso em 20 dez. 2019.

4.2.2 Máquina de Vetores Suporte

Ao contrário da RN, a Máquina de Vetores Suporte se baseia não na natureza, mas no senso comum. Ela define que o plano (que separa duas classes) de maior margem é o plano ótimo. O treinamento da SVM consiste, então, em encontrar esse plano.

Para problemas não linearmente separáveis, não há um hiperplano que separe as classes. Então o algoritmo consiste em definir o plano ótimo, estender esse plano para problemas não linearmente separáveis e mapear os dados para uma dimensão mais alta, onde talvez seja possível tornar um problema não linearmente separável em separável e achar o hiperplano. A equação do plano ótimo é definida por 4.10.

$$w \cdot x + b = 0 \tag{4.10}$$

O treinamento da SVM, baseado em encontrar w e b exibidos em 4.10, necessita de uma restrição para que a otimização ocorra numa janela entre as classes, definida em 4.11.

$$y_t(w \cdot x_t + b) \geq 1, \forall x_t \tag{4.11}$$

A obtenção do plano da SVM não depende do tamanho da base. Só depende dos vetores suporte, que são os que estão nos pontos paralelos ao plano ótimo. Eles são os que tocam a margem, como é possível visualizar na Figura 6. Diferente da Rede Neural, que pode ficar presa em um mínimo local, o treinamento da SVM é sempre ótimo - para problemas linearmente separáveis e factíveis.

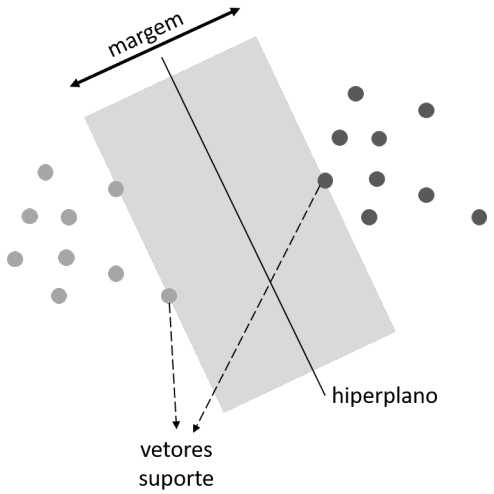


Figura 6 – Esquematização da SVM

Para problemas não factíveis, é possível utilizar uma variável de folga. Ela suaviza a restrição, permitindo erros para que seja possível encontrar a solução para problemas não factíveis. Esses casos de margem suave sempre têm solução, mesmo que o problema não seja linearmente separável, pois a SVM permitirá erros.

A SVM manda os dados originais para o hiperespaço para tentar encontrar um hiperplano que seja linearmente separável. Esse mapeamento pode ser feito através da função de *kernel*, sem precisar calcular o mapeamento, que é computacionalmente custoso. A Figura 7 explana como um conjunto de dados não linearmente separável no plano $2D$ pode se tornar linearmente separável no plano $3D$, tornando factível o encontro da fronteira de decisão entre classes (no caso, círculos verdes e quadrados vermelhos). Por conseguinte, tendo sido definido o espaço no qual cada classe está contida, é possível realizar a classificação dos dados. Depois de treinada, para predizer uma instância, a Rede Neural só precisa ser alimentada. Já na SVM, é preciso de todos os vetores suporte, portanto a classificação é mais complexa.

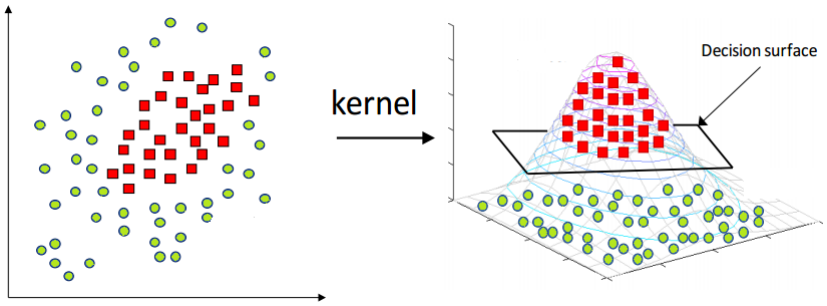


Figura 7 – Exemplo de Mapeamento de Hiperplano da SVM
Fonte: (JAIN, 2017)

A SVM não é um algoritmo inerentemente capaz de classificar multi-classes, sua estratégia é geral para classificação binária. No entanto, as estratégias “um-contra-todos” e “todos-contra-todos” (ALMEIDA, 2010) permitem a sua utilização em problemas com várias classes. Na primeira são geradas k SVMs, sendo k a quantidade de classes. Para cada uma das SVMs uma classe é fixada como positiva e as outras como negativas. Após k execuções, a classe mais votada será a classificação final da instância. Na segunda, são geradas $k(k-1)/2$ SVMs, é feita uma combinação 2 a 2 e, em seguida, a votação.

4.2.3 Comitês de Classificadores

Devido ao fato de seres humanos não serem capazes de aprender todo o conhecimento do mundo, procuramos nos tornar especialistas em determinadas áreas. E é comum especialistas de diversas áreas reunirem-se para discutir certos conteúdos. Analogamente, algoritmos também são especialistas em partes diferentes de um pro-

blema. “Comitês de classificadores são sistemas de classificação compostos por um conjunto de sistemas de classificação e por um método de combinação dos resultados” (SILVA; SANTOS, 2017).

Quanto mais complexo um problema, mas difícil é para um único algoritmo conseguir generalizar sobre ele. A tendência é que ele generalize demais - não distinguindo classes que na verdade são distintas - ou que se torne especialista em uma parte específica do problema. Surgem, então, os Comitês de Classificadores, com o objetivo de fazer com que uma combinação de algoritmos acerte instâncias que os classificadores individuais errariam. Ao diluir o viés de classificadores individuais, ou seja, reduzir a sensibilidade de um classificador individual ao conjunto de treinamento, aumenta-se o desempenho de processamento. Mas, por outro lado, há como desvantagem a necessidade de treinar vários algoritmos, e a interpretação deles fica mais complexa.

Ao selecionar um conjunto de classificadores para o Comitê, deve-se ter em foco a diversidade, pois utilizar classificadores iguais é apenas perda de recurso. Os algoritmos que compõem o classificador podem ser os mesmos, mas seus hiperparâmetros devem ser diferentes. No entanto, para deixar o sistema mais genérico e flexível a diferentes problemas, pode-se fixar o algoritmo e os hiperparâmetros e variar o conjunto de treinamento.

Existem diferentes técnicas para criação de Comitês especialistas: *Bagging, Boosting, Stacking, Random Forest, Gradient Boosting*, entre outros. Proposta por (BREIMAN, 1996), a técnica de *Bagging*, de implementação simples e intuitiva, obtém a diversidade a partir de reamostragens do conjunto de dados original. Esses subconjuntos são independentes e criados aleatoriamente com reposição. O algoritmo é fixo e cada subconjunto é treinado pelo algoritmo. Porém, o contrário também é válido: fixar o conjunto de dados e variar os parâmetros do algoritmo.

A classificação final é dada a partir da combinação dos classificadores por voto majoritário ou por um somatório das opiniões. Caso a classe seja definida pela resposta aprovada pela maioria dos classificadores, elas têm peso igualitário. No somatório, que pode ser usado em classificações contínuas, a confiança sobre a resposta é elevada ao utilizar o Comitê. Essa operação de classificação é ilustrada na Figura 8, na qual é possível notar que três classificadores passam pelo processo de aprendizado independentemente e, por fim, suas “opiniões” são combinadas para formar a decisão final do Comitê.

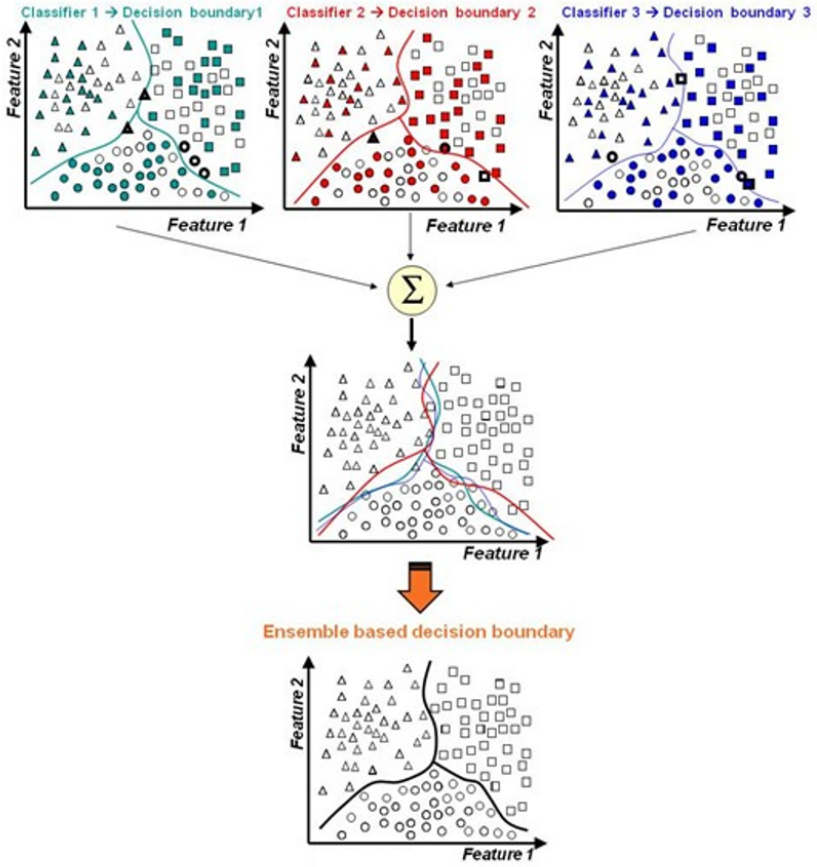


Figura 8 – Exemplo de Comitê de Classificadores
Fonte: (POLIKAR, 2006)

4.3 Técnicas para Validação e Avaliação do Classificador

Uma vez que o modelo de AM foi treinado, ele precisa ser validado, para garantir a sua capacidade de generalização sobre novos dados. Afinal, o objetivo final de se ter um modelo é poder utilizá-lo em um sistema para predizer novas instâncias que surgirão no dia-a-dia do público alvo do problema. É comum que as grandes bases de dados sejam divididas em um conjunto para treinamento e outro para testes. Dessa maneira, garante-se que após o treinamento o modelo será testado com instâncias nunca antes vistas. Essa validação irá garantir que o modelo não sofreu *overfitting*, ou seja, que ele não "decorou" o conjunto de dados apresentados ao invés de realmente aprender.

No entanto, nem sempre há disponibilidade de bases de dados grandes o suficiente para que possam ser divididas em dois conjuntos distintos. Nesses casos, pode-se fazer uso da estatística para garantir a capacidade de generalização do modelo através de uma técnica chamada *Cross-validation*.

Tendo sido feito o treinamento e a validação dos modelos, a partir do momento em que se compara modelos de aprendizado de máquina, é necessário a utilização

de métricas para quantificar os resultados e permitir uma análise qualitativa. Existem diferentes métricas para serem utilizadas em diferentes casos, desde aprendizado não supervisionado até o supervisionado.

A seguir, será descrita a técnica de *K-fold Cross-validation* e algumas métricas para avaliação de modelos.

4.3.1 *K-fold Cross-Validation*

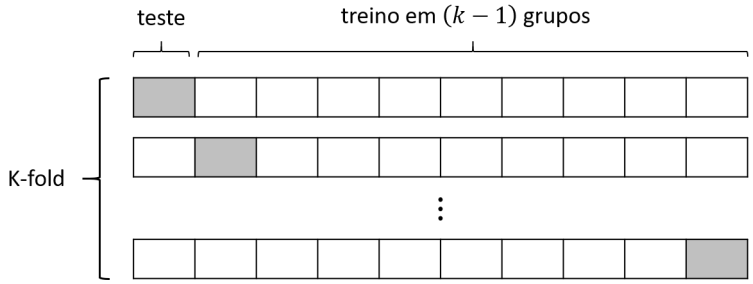


Figura 9 – Esquematização do *K-fold Cross-validation*

O método *k-fold Cross-validation* consiste em dividir o conjunto de dados em k partes, para que uma dessas partes seja usada para testes e o restante para treinamento dos modelos. *Folds* são conjuntos disjuntos, quantificados por k, amostrados de maneira estratificada (mantendo a distribuição das classes). Convencionalmente utiliza-se $k = 10$.

A Validação Cruzada consiste em executar o treinamento e o teste k vezes. Em cada iteração, os *folds* de teste e treino se alternam e o erro vai sendo computado de acordo com a métrica de avaliação escolhida. Ao final do processo haverá k medidas de avaliação, com as quais se calcula a média e o desvio-padrão. A execução do *k-fold* deve ser feita para cada combinação de hiper-parâmetros do algoritmo.

A Figura 9 ilustra o processo do método, tendo sido definido $k = 10$. É aleatoriamente selecionado um grupo para teste, em cinza, e ele vai sendo alternado em cada rodada da validação. Os grupos restantes, em branco, compõem o conjunto de treinamento. Este método também pode ser dividido em 3 conjuntos distintos, para treino, teste e validação de treinamento.

4.3.2 Métricas para avaliação

As métricas visam medição do desempenho dos algoritmos de AM. A seguir, serão descritas algumas das principais medidas utilizadas na literatura. Um acerto acontece quando uma instância é classificada como pertencente à sua verdadeira classe. O oposto acontece na contabilização de erros.

- Acurácia: Razão entre quantidade de acertos e o total de instâncias, podendo ser de treinamento, validação e teste. Caso o conjunto de dados seja desbalanceado, essa métrica perde seu significado, pois ela pode ser alta apenas porque a classe dominante foi corretamente classificada.
- Taxa de Erro: Razão entre quantidade de erros e o total de instâncias, podendo ser de treinamento, validação e teste.
- Precisão: Razão entre as instâncias classificadas corretamente (verdadeiros positivos) e o somatório das instâncias classificadas corretamente com as instâncias preditas como pertencentes à classe, mas que na verdade não pertencem (falsos positivos). Indica o quão puro é o algoritmo.
- Cobertura: Razão entre as instâncias classificadas corretamente (verdadeiros positivos) e o somatório das instâncias classificadas corretamente com as instâncias que pertenciam àquela classe, mas que foram preditas como pertencentes a outras (falsos negativos). Indica a quantidade de acertos que o algoritmo conseguiu fazer para determinada classe.
- *F-measure*: Média Harmônica entre precisão e cobertura, definida em 4.12.

$$2 \times \frac{(\text{Precisão} \times \text{Cobertura})}{(\text{Precisão} + \text{Cobertura})} \tag{4.12}$$

Outro artifício bastante utilizado para avaliação dos modelos é a Matriz de Confusão. Apesar de não ser uma métrica unitária, ela é uma tabela que mostra as frequências de classificação para cada classe do modelo. Ela exibe todos os casos de verdadeiros positivos, verdadeiros negativos, falsos negativos e falsos positivos. Os valores que compõem esse tipo de matriz são utilizados na contabilização das métricas citadas anteriormente, portanto sua construção facilita o cálculo das métricas.

5 Classificação Textual para Predição de Esforço de Software

No capítulo anterior, foram descritas técnicas de pré-processamento necessárias para se trabalhar com o aprendizado de máquina sobre componentes textuais. A Figura 10 mostra o passo-a-passo completo para execução de classificadores em documentos de texto. Esse será o *pipeline* utilizado nos experimentos deste trabalho, sendo o conjunto de dados formado por Relatórios de Erros de um *software* e o pré-processamento composto por tokenização, remoção de ruído e lematização. A etapa seguinte, de extração de características será aqui analisada com mais profundidade.

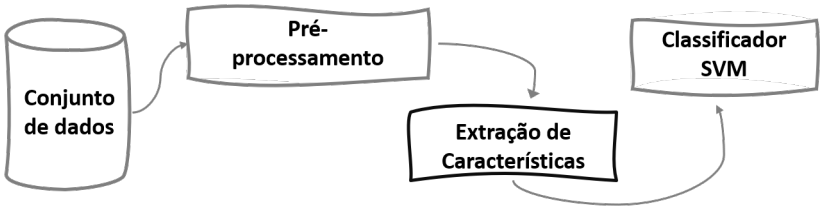


Figura 10 – Esquematização Genérica do *Pipeline*

O conteúdo de texto lematizado - um vetor de termos resultante do *pipeline* de pré-processamento - ainda não é compreensível por intérpretes de linguagem formal, ainda não é passível de interpretação pelos algoritmos. A extração de características irá, por conseguinte, fazer uma transformação final dos textos em vetores numéricos. Os textos perderão seu sentido semântico aos olhos humanos, mas passarão a ser compreendidos pelos modelos matemáticos. Existem algumas formas distintas de executar esse processo, como BoW (HARRIS, 1954), TF-IDF (MANNING; RAGHAVAN; SCHüTZE, 2009), PV-DM (LE; MIKOLOV, 2014), entre outas.

O presente trabalho realiza uma comparação dos métodos BoW e PV-DM, com a proposição de um terceiro método disposto por uma combinação de PV-DMs. A seguir, serão descritas as técnicas, seus diferenciais e suas vantagens ou desvantagens.

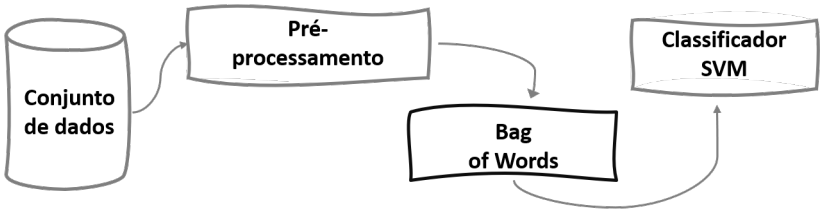


Figura 11 – Esquematização da Abordagem 1

5.1 Abordagem 1: *Bag of Words*

A primeira abordagem, como demonstrado na Figura 11, utiliza-se do método *Bag of Words*. Literalmente um "Saco de Palavras", a técnica é uma representação simplificada de documentos, que revela a frequência de ocorrência dos termos. Tomando como exemplo um documento formado pelas duas frases a seguir e seus vetores de lematização, é gerado um vocabulário com o conjunto das palavras existentes no documento. Com o vocabulário construído, contabiliza-se a ocorrência dos termos no documento que será representada num formato de dicionário {*"chave":valor*}. Sendo a chave uma representação para os termos, e o valor a quantidade de ocorrências no documento.

```
Sentença 1: Apesar de o sistema móvel estar funcionando,
ele deve funcionar em múltiplas plataformas.
Lematização 1: ["sistema", "móvel", "estar", "funcionar",
"dever", "funcionar", "múltipla", "plataforma"]

Sentença 2: O sistema web funciona?
Lematização 2: ["sistema", "web", "funcionar"]

Vocabulário: ["sistema", "móvel", "estar", "funcionar", "dever",
"múltipla", "plataforma", "web"]

Contagem: {"sistema":2, "móvel":1, "estar":1, "funcionar":3,
"dever":1, "múltipla":1, "plataforma":1, "web":1}
```

Nos Relatórios de Erros, cada problema contém sua descrição e seus Pontos de História, representando uma instância. As sentenças do exemplo apresentado poderiam constar na descrição de um problema sobre um sistema que deveria funcionar em múltiplas plataformas, mas não está funcionando na plataforma *web*. Esse documento teria, portanto, um valor de Ponto de História associado para estimar sua complexidade de desenvolvimento. Na utilização do BoW, tendo sido construída a estrutura de contagem dos termos, cada instância será transformada em vetor no formato de *one-hot encoding*. Ou seja, todo o documento será representado por uma grande matriz em que cada elemento corresponde a uma instância do conjunto de dados. Essa instância será um vetor numérico cujas colunas correspondem ao vocabulário formado. Então, caso a determinada instância possua o termo, o elemento do vetor será equivalente à quantidade de ocorrências, sendo 0 caso a instância não contenha o termo. A Figura 12 exibe o vetor concebido a partir do exemplo anterior. Em seguida, esse vetor é passado como entrada para o classificador.

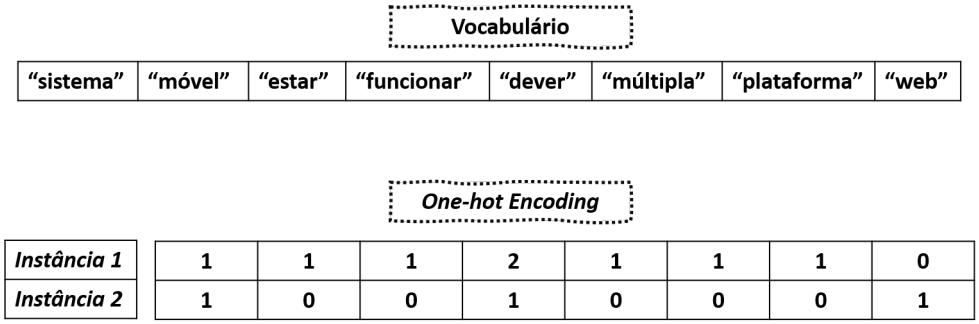


Figura 12 – Representação do BoW

Apesar de simples, essa bordagem apresenta uma série de desvantagens. O BoW, como expõem (WU; HOI; YU, 2010), apresenta uma limitação cítica em relação à perda de informação semântica. Para o BoW, apenas a ocorrência do termo é levada em consideração; seu significado, contexto e ordem de aparecimento são ignorados. Dessa forma, documentos escritos de maneiras similares terão vetores similares.

A desconsideração do contexto é um problema devido às imensas possibilidades das linguagens. Palavras iguais podem ter significados completamente diferentes de acordo, apenas, com o contexto em que são utilizadas. Então o aprendizado pode vir a ser comprometido por essa razão. Outro grande problema é o tamanho do vetor de contagem. Principalmente em grandes documentos, o tamanho do vetor pode ser enorme, o que irá requisitar muito mais poder de processamento e tempo. Não somente, o classificador pode se tornar vulnerável a *overfitting* e perder sua capacidade de generalização para novos conjuntos de dados. Essa situação é bastante comum no uso de Redes Neurais, pois vetores cujos tamanhos se encontram na casa dos milhares significam redes com milhares de parâmetros para serem treinados e regulados.

5.2 Abordagem 2: *Paragraph Vector - Distributed Memory*

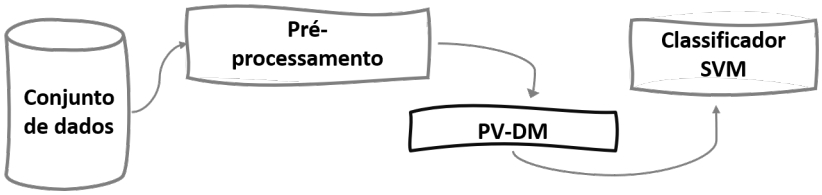


Figura 13 – Esquematização da Abordagem 2

A segunda abordagem deste trabalho (Figura 13) utilizar-se-á do *Paragraph Vector - Distributed Memory*. O PV-DM, de acordo com (LE; MIKOLOV, 2014), é um *framework* não supervisionado, eficaz em vários domínios. Sua implementação é baseada em uma Rede Neural MLP, através da qual o algoritmo aprende a distribuição contínua de vetores de tamanho fixo que representam textos de tamanhos variados.

Textos que seriam representados em vetores de alta dimensão com o BoW podem ser transformados em vetores de baixa dimensão com o PV-DM.

Consoante (SOARES, 2018),

> O treinamento do PV-DM assume que os vetores de palavras contribuem na predição da próxima palavra em uma sentença. Os vetores de palavras codificam a semântica dos textos como um subproduto dessa predição. Cada documento é denotado como uma única coluna em uma matriz de documentos D, e cada termo é representado por uma única coluna em uma matriz de termos W. O parágrafo e o vetor de termos são combinados para predizer o próximo termo no contexto de cada texto. O vetor de documentos representa uma memória que ajuda a indicar as partes faltantes do contexto atual. O tamanho de um contexto é fixo, e suas palavras são amostradas de uma janela deslizante sobre o parágrafo. O PV-DM emprega o Gradiente Descendente via *backpropagation*. A cada época, o PV-DM amostra um contexto de tamanho fixo de um parágrafo aleatório, calcula o erro gradiente e o utiliza para atualizar os pesos da rede.

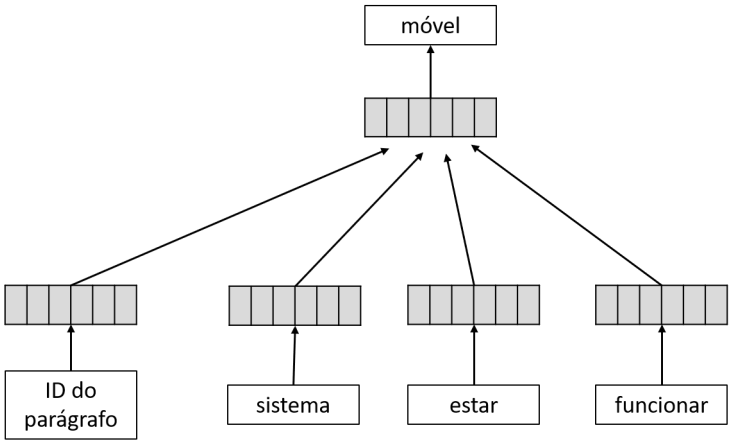


Figura 14 – Esquematização do PV-DM

A Figura 14 ilustra as matrizes D e W citadas por (SOARES, 2018). A Figura 15 exibe o funcionamento das janelas deslizantes, considerando seu tamanho igual a 2. À medida em que a janela passa pelos termos, cria-se o contexto para aquele parágrafo. A Rede irá, então, aprender o contexto pela quantidade de vezes que ele aparece. Ou seja, a quantidade de ocorrências do par **(sistema, móvel)**, graças ao seu significado semântico, deve ser muito maior do que de **(sistema, cachorro)**. Assim, dada como entrada o termo **sistema**, a probabilidade da saída ser **móvel** ou ***web*** será muito maior do que **cachorro**.

Similar ao BoW, é necessário um vetor de vocabulário para que construa-se uma representação *one-hot encoding*. A diferença é que no PV-DM cada palavra será

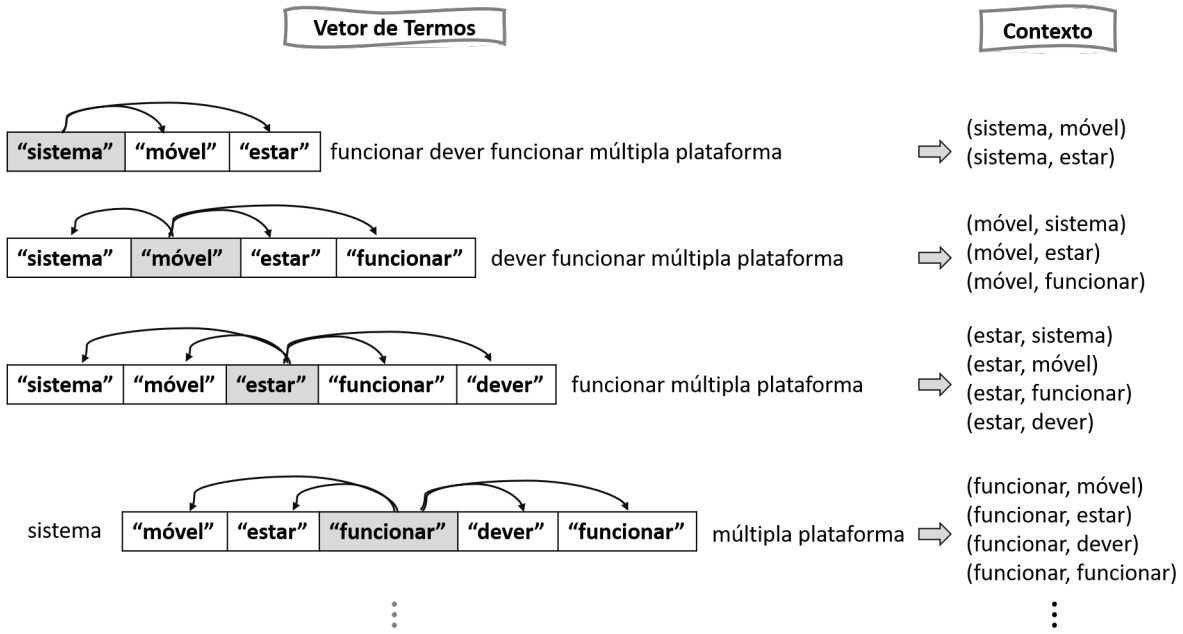


Figura 15 – Representação das janelas deslizantes

formada por um vetor do tamanho do vocabulário, contendo 1 na posição referente à palavra escolhida, e 0 em todas as outras posições. Este vetor será a entrada da Rede (última linha da Figura 14, composta pelo ID do parágrafo e os termos “sistema”, “estar”, “funcionar” em forma de vetores). A saída da RN (primeira linha da Figura 14) também é um vetor em que cada posição representa uma palavra do vocabulário, tendo como valor a probabilidade daquela palavra ser a palavra seguinte à entrada. (JúNIOR, 2019) reitera que se as palavras têm contextos semelhantes, então a Rede estará motivada a aprender vetores de palavras similares. As palavras tendem a ter contexto semelhantes justamente quando têm significados parecidos, quando são sinônimos ou quando estão relacionadas de alguma forma. Dessa maneira, o vetor do termo “sistema” tende a ser mais próximo do vetor de “móvel” do que de “cachorro”. Esses vetores, encapsulados nos pesos da Rede, representam os termos do vocabulário e carregam informação semântica, já que existe uma noção de “distância” entre os termos, que varia com o contexto. Esses vetores representam a tradução da linguagem não formal para a formal, são as características as quais desejava-se extrair dos textos para que se tornassem compreensíveis pelos algoritmos.

Em resumo, apesar do PV-DM ser treinado para calcular a probabilidade da próxima palavra em um determinado contexto, sua aplicação, de fato, está nos pesos ajustados. Esses pesos, após o treinamento, são extraídos para compor o vetor de palavras que será posteriormente aprendido pelo classificador SVM do *pipeline*.

5.3 Abordagem 3: Comitê de PV-DM

Para a terceira abordagem, propõe-se não apenas o uso do PV-DM, mas de um Comitê de PV-DM. (SILVA; SANTOS, 2017) apresentam o crescente uso de Comitês de Classificadores, em diversas tarefas distintas, com o intuito de obter sistemas cada vez mais eficazes: classificação para detecção de intrusão em redes de computadores (RAMOS; SANTOS, 2015), predição de rendimento escolar (NOGUEIRA, 2015), detecção de opinião em mensagens curtas (LOCHTER; ZANETTI; ALMEIDA, 2016), entre outros.

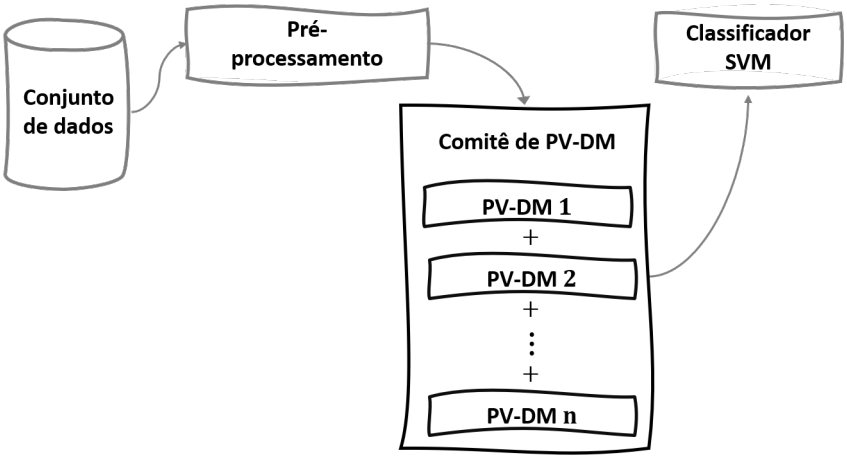


Figura 16 – Esquematização da Abordagem 3

Como já abordado no capítulo anterior, o uso de Comitês é bastante interessante para problemas complexos, que é o caso do Processamento de Linguagem Natural. Apesar disso, não foi visto na literatura a utilização de Comitê de PV-DM para classificação textual. Por isso, a proposta da abordagem 3 será executada e comparada com as abordagens anteriores.

Conforme (KUNCHEVA, 2014), Comitês são construídos distribuindo diferentes bases de dados aos classificadores, ou a mesma base com diferentes conjuntos de atributos, ou construídos por uma combinação de todos o métodos. A metodologia aqui proposta, exposta da Figura 16, não fará distribuição da base de dados, mas sim de parâmetros do PV-DM. Cada PV-DM base - selecionado de 1 a n - será treinado com todo o conjunto de dados, e seus pesos serão somados aos pesos pós-treinamento do classificador seguinte para que ao fim seja obtido um modelo mais confiável e preciso. A quantidade de PV-DMs executada, definida na Figura 16 por n, é um hiper-parâmetro do Comitê cujo valor pode ser fixo ou variado para busca de melhores resultados. Cada PV-DM inerentemente diverge entre si devido à sua inicialização dos pesos com valores aleatórios, uma característica intrínseca à Rede Neural. Além disso, o tamanho das janelas é um hiper-parâmetro do PV-DM que irá definir o tamanho do contexto, e, portanto, também podem ser variados. Vale destacar que os pesos treinados extraídos

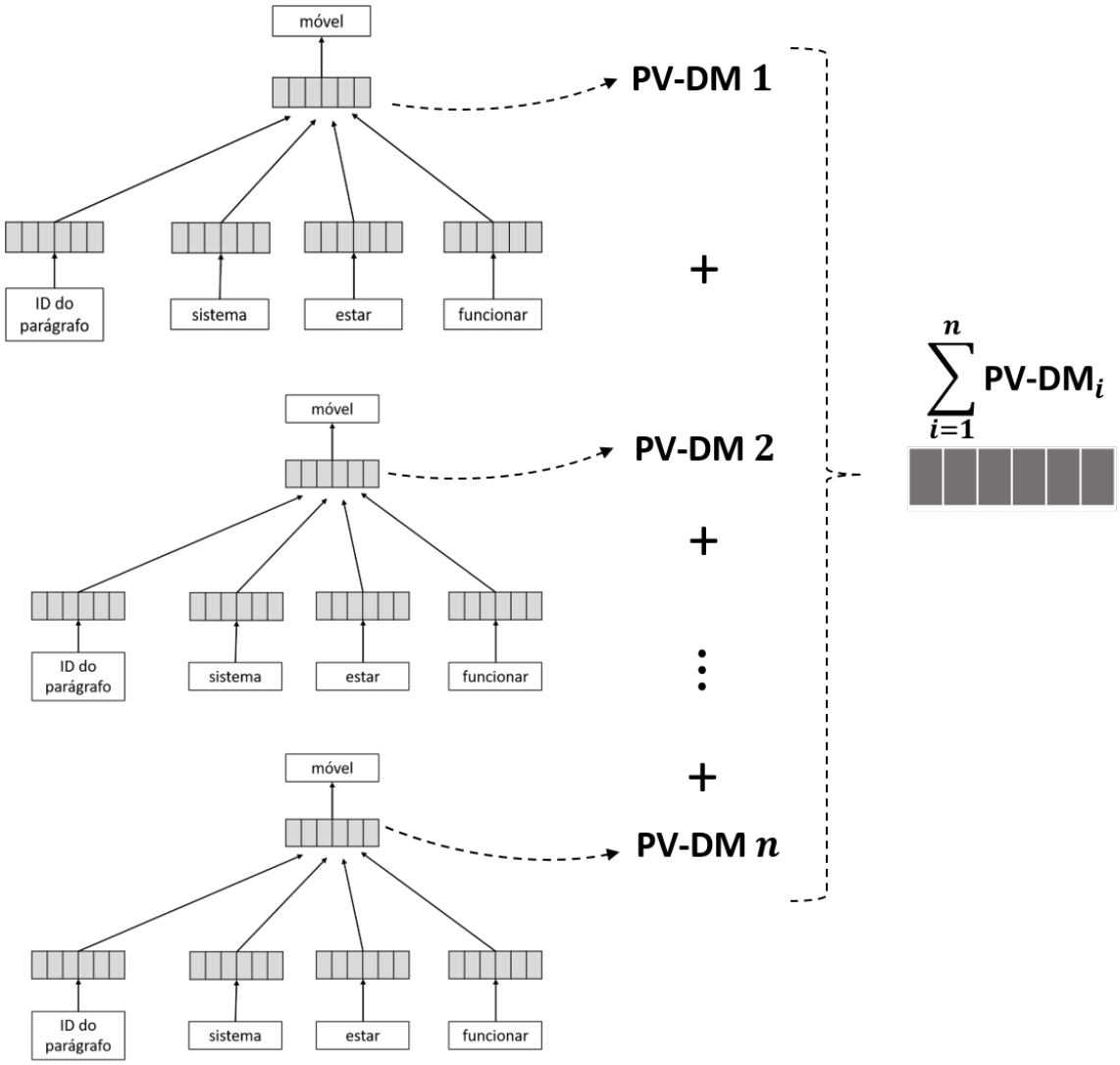


Figura 17 – Esquematização do Comitê de PV-DM

de cada PV-DM correspondem ao vetor de extração de características dos termos, que servirá como entrada para o classificador SVM.

Para exemplificar, a Figura 17 ilustra a execução dos n PV-DMs e a extração de seus pesos que, ao serem somados, geram um vetor de representação dos termos mais confiável (em cinza mais escuro) do que o vetor gerado por um único PV-DM (cinza claro). Apesar de o conjunto de dados de entrada ser o mesmo, os pesos de inicialização da Rede Neural são iniciados aleatoriamente, de forma que cada vetor extraído será diferente a cada execução do PV-DM.

6 Experimentos

Consoante (MARTIN, 2005), Pontos de História formam uma métrica subjetiva utilizada por times ágeis para estimar o esforço necessário para cumprir uma tarefa de desenvolvimento. Esta seção irá descrever a base de dados de Pontos de História utilizada e a análise experimental.

6.1 Base de Dados

Visto que este trabalho objetiva o aprendizado da experiência humana, através de algoritmos de IA, para a estimação de esforço na correção de *software*, foram utilizados Relatórios de Problemas para compor a base de dados. Os relatórios empregados são oriundos do projeto *Aptana Studio*[1] (APSTUD). O APSTUD é um projeto *opensource* (*software* cujo código-fonte pode ser livremente utilizado e adaptado) para criação de um ambiente integrado de desenvolvimento (IDE, do inglês *Integrated Development Environment*) para plataformas *web*.

Times de desenvolvimento ágil comumente utilizam-se de sistemas rastreadores de problemas como *Jira*[2], *Trac*[3], *Redmine*[4], *Asana*[5], etc. Esses sistemas fornecem rótulos para gravar e acompanhar o progresso de todos os problemas identificados pelo usuário até a correção deles. Reporte de falhas e solicitações de melhorias técnicas ou de funcionalidades já existentes no produto em manutenção caracterizam-se como problemas reportados na ferramenta. As novas solicitações podem ser oriundas de clientes, usuários ou de engenheiros de teste.

O APSTUD tem seu processo de desenvolvimento descrito no *Jira*. Nessa ferramenta, cada Relatório de Problema possui uma infinidade de campos para descrevê-lo, como título, resumo, descrição, *status*, nome do projeto, prioridade, pessoa responsável, prazo, ambiente, estimativa de tempo, tempo gasto, etc., além de outros campos customizáveis. Um dos campos customizáveis do APSTUD é o de Pontos de História, mantido e atualizado no decorrer do projeto. A Figura 18 demonstra um problema como exemplo, com parte de seus campos na ferramenta rastreadora.

Muitos campos não são obrigatórios, e muitos encontram-se vazios ou possuem conteúdo irrelevante para o objetivo deste trabalho. Então, para realização dos expe-

[1] http://www.aptana.com/
[2] https://www.atlassian.com/software/jira
[3] https://trac.edgewall.org/
[4] https://www.redmine.org/
[5] https://asana.com/

rimentos, foram extraídos os campos de descrição dos problemas e o campo customizado de Pontos de História dos relatórios. As instâncias selecionadas foram de problemas já finalizados e corrigidos, para garantir que o Ponto de História definido tenha sido o último atribuído, correspondente à estimativa de complexidade verdadeiramente adequada para o desenvolvimento da atividade de correção.

A base original contém 538 problemas, dos quais 46 foram removidos por terem seus campos de descrição vazios. Assim, o conjunto de dados utilizados neste trabalho possui 2 atributos e 492 instâncias. O atributo de descrição está em inglês e pode conter detalhes do erro ocorrido, trechos de código responsáveis ou possivelmente responsáveis pelo problema, trechos de *log*, datalhes do ambiente no qual o *software* estava sendo executado e a correção esperada. O atributo de Pontos de História define o esforço para correção da tarefa, definido previamente durante o *Planning Poker*. Ele é o rótulo que diz à qual classe a instância pertence, sendo um elemento do seguinte conjunto: $\{x | x \in \{0, 1, 2, 3, 5, 8, 13, 20, 40, 89\}\}$.

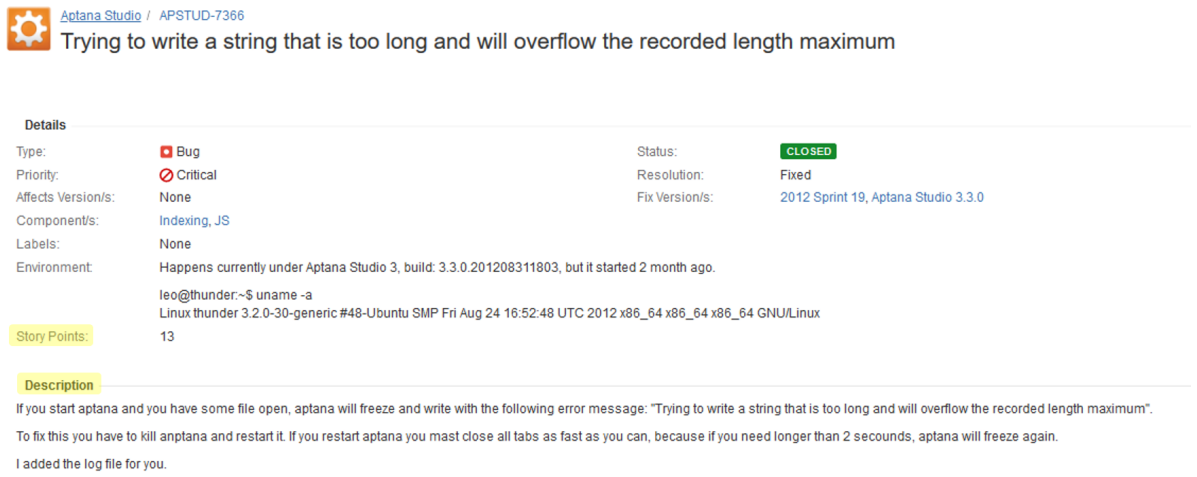
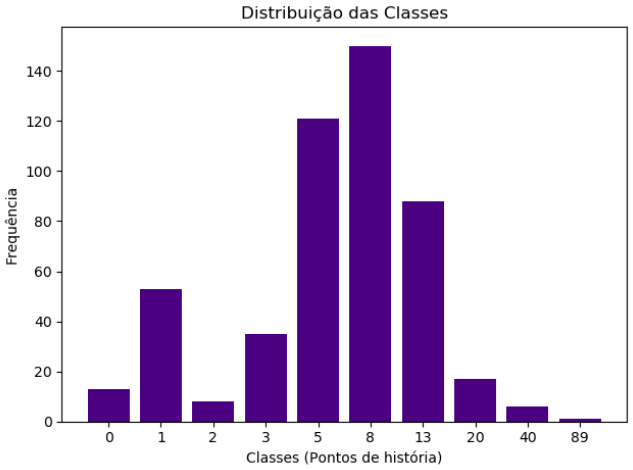


Figura 18 – Exemplo de um problema finalizado na plataforma *Jira*
Disponível em: <https://jira.appcelerator.org/browse/APSTUD-7366>
Acesso em 30 nov. 2019
Grifo da autora em amarelo para destacar os atributos utilizados



(a) Distribuição de ocorrência das classes

(b) Nuvem de palavras de maior ocorrência nas descrições de problemas do APSTUD

Figura 19 – Ocorrência das classes e das palavras da APSTUD

O *Jira* permite ao usuário colocar nas descrições dos problemas trechos de texto com formatação diferenciada, para que certas exibições fiquem mais intuitivas e naturais na ferramenta. Esses trechos são delimitados por marcadores tais como *{code}, {quote}, {html}, {noformat}, {panel}*. Os trechos dentro desses marcadores, na grande maioria das vezes, são formados por linguagem formal, composta de símbolos e estruturas distintas à LN. Como o foco deste trabalho é o aprendizado de linguagem natural, esses trechos seriam considerados ruídos. Portanto, foram substituídos por um identificador no formato *tag_nome*, sendo *nome* o respectivo nome do referente marcador. Dessa forma, o algoritmo pode aprender que certas instâncias possuem um conteúdo diferente - possivelmente relevante para a classificação -, mas elas possuem um padrão de ocorrências no documento que não prejudicarão o aprendizado da LN. Tendo sido detalhado esse processo, pode-se explanar que ele foi realizado na etapa de remoção de ruído, junto com todos os passos do pré-processamento levantados no capítulo 4 (tokenização, remoção de ruído e lematização).

As Figuras 19a e 19b apresentam, respectivamente, a distribuição das classes na base de dados e os termos de maior ocorrência nas descrições dos problemas. Pode-se observar que as classes extremas (superiores e inferiores) têm bem menos presença no conjunto de dados do que as centrais, de valores entre 5 e 13. Na nuvem de palavras, quanto maior e mais colorida a palavra, maior a sua quantidade de ocorrências em relação às outras. Termos cuja frequência é muito rara não são representados na nuvem de palavras.

6.2 Seleção de Modelos

Os hiper-parâmetros da SVM foram otimizados por meio de uma busca randomizada (BERGSTRA; BENGIO, 2012). Cada combinação de hiper-parâmetros foi avaliada por meio de *Cross-validation* com *10-folds*. O intervalo de parâmetros foi replicado para todas as abordagens do trabalho. Para execução da Máquina de Vetores Suporte foi utilizada a implementação da biblioteca *Sckit-Learn*[6] para a linguagem de programação *Python*[7] *(versão 3.7)* .

Sobre a extração de características, na abordagem 1, para execução do BoW foi utilizada a classe *CountVectorizer*, também do *Sckit-Learn*. Sua execução não exige definição de hiper-parâmetros. Já na abordagem 2, o PV-DM teve sua janela fixa em 10 e tamanho de vetor em 100. Para a abordagem 3, o Comitê teve o número de aprendizes fixo em 100 PV-DMs, assim como o tamanho do vetor, mas a janela teve seu tamanho variado para busca do melhor resultado. Para execução do PV-DM nos experimentos foi utilizada a implementação *Doc2Vec* da biblioteca *Gensim*[8]. A Tabela 1 exibe os limites definidos para busca da SVM e do comitê de classificadores.

Distribuições de Probabilidade da SVM (Abordagens 1, 2 e 3)		
C	exponencial com $\lambda = 10^{-3}$	
Função de *kernel*	uniformemente discreto em {linear, *sigmoid*, RBF}	
γ	uniformemente em $[0, 1]$	
Grau	uniformemente discreto em $\{1, 2, 3, 4\}$	
Distribuições de Probabilidade do Comitê de Classificadores (Abordagem 3)		
Número de Aprendizes	fixo em 100	
Algoritmo: PV-DM	Tamanho do Vetor	fixo em 100
	Janela	uniformemente discreta em $\{1, \ldots, 10\}$

Tabela 1 – Distribuições de Probabilidade para a Busca Randomizada dos Hiper-parâmetros

6.3 Resultados

Desde que o conjunto de dados de treinamento estava desbalanceado, utilizou-se a *f-measure* como métrica de avaliação dos modelos gerados. Os resultados foram compilados na Tabela 2, com seus respectivos hiper-parâmetros geradores descritos

[6] https://scikit-learn.org/stable/
[7] https://www.python.org/
[8] https://radimrehurek.com/gensim/index.html

na Tabela 3. Pode-se notar que a segunda abordagem, com um único PV-DM, mostrou melhores resultados, com *f-measure* de aproximadamente 93%.

	Abordagem 1 BoW	Abordagem 2 PV-DM	Abordagem 3 Comitê de PV-DM
Classificador SVM	$0,292683$	$0,932927$	$0,796748$

Tabela 2 – *F-measure* da SVM nas diferentes abordagens

As Figuras 20 e 21 ilustram as matrizes de confusão das três abordagens. A maldição da dimensionalidade (RAUDYS; JAIN, 1991) é clara, já que o tamanho do vocabulário passa dos 10.000 termos, e o número de instâncias por classe é pequeno, mesmo nas de maiores números de casos. De acordo com o autor, o número de instâncias necessárias para o Aprendizado de Máquina de problemas complexos cresce exponencialmente com o número de atributos (ou dimensões). A SVM utiliza-se da distância entre os vetores de suporte e o hiperplano para definir o plano ótimo. Em casos com poucas dimensões, a distância euclidiana, expandida em 6.1 com n dimensões, funciona bem. No entanto, quando a quantidade de dimensões está na casa das centenas ou milhares, dimensões muito grandes podem se diluir em meio às muitas outras dimensões, de maneira que a distância perde seu valor e seu real significado.

$$d = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2 + \cdots + (n_a - n_b)^2} \tag{6.1}$$

É evidente a incapacidade da abordagem 1 em aprender bem o problema. Essa abordagem, por gerar um vetor de maior dimensão, sofre mais durante o aprendizado. Analisando sua Matriz de Confusão, esse método conseguiu realizar predições em todas as classes, exceto a 89, que é realmente mais difícil, pois só contém uma única instância. Apesar de ter classificado algumas instâncias como pertencentes ao valor

	Abordagem 1 BoW	Abordagem 2 PV-DM	Abordagem 3 Comitê de PV-DM
C	$506,8162186073297$	$72,96958817266247$	$782,412143494269$
Função de *kernel*	*sigmoid*	RBF	linear
γ	$0,1613879925068219$	$0,05289029599731742$	$0,6198370249260736$
Grau	2	1	1

Tabela 3 – Hiper-parâmetros responsáveis pelos melhores resultados da SVM por abordagem

2, não houve nenhuma ocorrência de verdadeiros positivos nessa classe. O método parece ter sofrido *underfitting*, não tendo se ajustado bem aos dados.

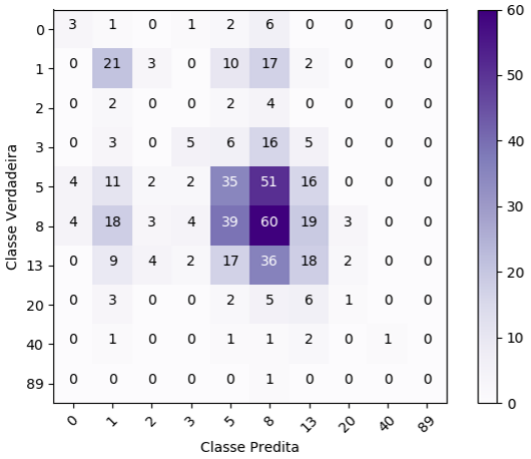
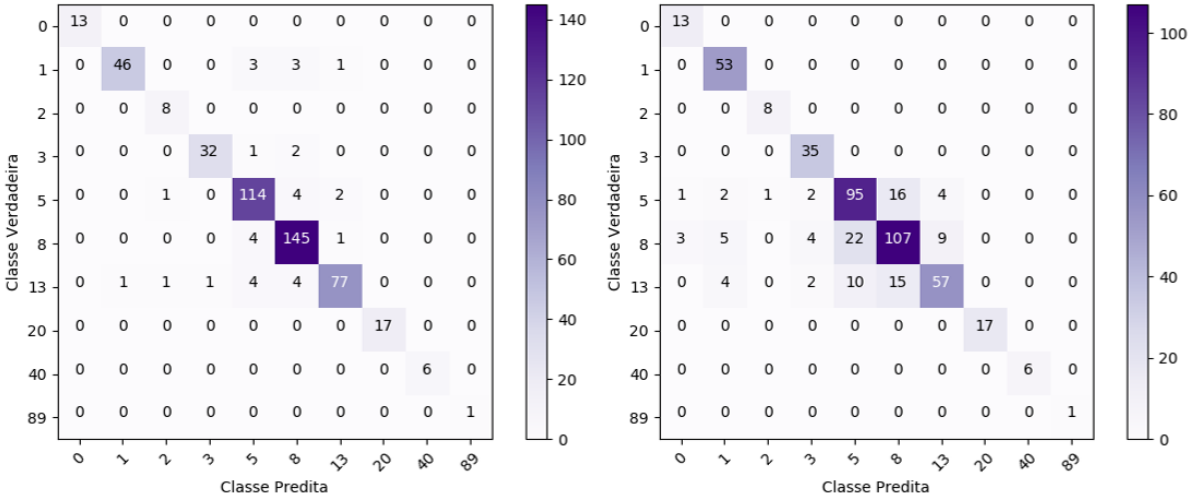


Figura 20 – Matriz de Confusão - Abordagem 1

Observa-se uma grande dificuldade entre as classes $5, 8$ e 13, sendo o local onde se observa maior confusão na predição. Essa confusão se estende à classe 1, que teve mais falsos positivos nas três classes mais problemáticas do que verdadeiros positivos. As ocorrências de falsos negativos também foram razoáveis e maiores que as de verdadeiros positivos. De fato, a fronteira entre as classes se mostra complexa e irregular, de difícil aprendizado. Principalmente entre as classes já citadas, $5, 8$ e 13.



(a) Abordagem 2 (b) Abordagem 3

Figura 21 – Matriz de Confusão - Abordagens 2 e 3

O uso de uma Rede Neural - o PV-DM - na extração de características mostrou surtir grandes efeitos na predição dos Pontos de História. A segunda abordagem apresentou um resultado muito bom, com cerca de 93% de *f-measure*. Por outro lado,

(JúNIOR, 2019) apresentou resultados ruins para com os títulos da APSTUD ao utilizar o PV-DM. O autor justifica seu resultado à limitação dos títulos de relatórios disponíveis, já que a técnica PV-DM é totalmente dependente de pareamentos de palavras que são formados pela quantidade de palavras que possui cada relatório. Aqui, além de ter sido utilizado o campo de descrição ao invés do título, que possui mais detalhes e maior quantidade de termos, também foi realizada a identificação dos trechos de marcadores especiais, como códigos e citações, que parecem ter trazido impactos positivos nos resultados.

Esse modelo apresenta um ajustamento adequado aos dados, tendo classificações corretas em todas as classes do problema. A sua Matriz de Confusão - Figura 21a - apresenta quase que uma diagonal perfeita, com alguns leves deslizes nas mesmas classes problemáticas à metodologia anterior, $5, 8$ e 13. As ocorrências de falsos positivos neste caso foram superiores às de falsos negativos. A grande concentração destes casos está na subestimação de atividades que deveriam ser classificadas com Ponto de História 13, mas que foram classificadas com complexidade inferior - entre 1 e 5 -, o que é um ponto negativo em vista das grandes chances de multas e quebra de contrato por ultrapassar prazos estipulados.

A Tabela 3 expõe que o hiper-parâmetro C foi bem mais baixo na segunda abordagem, comparado às outras. Visto que esse parâmetro controla o quanto de erros são permitidos durante o treinamento, a suavização dessa variável, ao permitir mais erros, possibilitou um melhor ajuste aos dados, já que margens mais largas se tornaram factíveis. Fica claro que a primeira abordagem, de maior valor neste parâmetro, não conseguiu o ajuste ideal com a sua margem mais rígida. A utilização do *kernel* RBF, que permite mapeamentos não lineares, na abordagem 2 também conta como fator positivo na captura da complexidade dos dados.

A partir das melhorias obtidas no uso do PV-DM, esperava-se obter resultados ainda melhores na terceira abordagem, com a utilização de Comitês de PV-DM. Na prática, o cenário foi um pouco diferente. A proposta apresentou um bom resultado - cerca de 80% de *f-measure* -, porém ainda inferior à utilização de um único PV-DM. Na sua Matriz de Confusão, Figura 21b, é possível notar um comportamento similar à abordagem anterior, porém com mais ocorrências de falsos positivos e falsos negativos nas mesmas classes $5, 8$ e 13. Apesar disso, essa técnica também foi capaz de classificar instâncias de todas as classes, tendo, inclusive, maior facilidade na distinção das classes 1 e 3 do que a abordagem vencedora. É possível que uma busca randomizada de maior orçamento seja capaz de trazer melhores resultados nesse caso, pois é visto que os especialistas distinguiram melhor casos mais simples, mas ainda tiveram dificuldade em classes mais complexas, cujas fronteiras são mais próximas.

7 Conclusões

A predição de esforço para construção ou correção de *software* é uma área importante na Engenharia de *Software*. (CHARETTE, 2005; OLIVEIRA, 2006) avaliam que, no início dos anos 2000, estimativas de esforço de projetos de *software* ineficazes tiveram um impacto na economia dos Estados Unidos de algum valor entre 25 e 75 bilhões de dólares. O grande impacto financeiro decorre de razões como atraso nos prazos, perda de vantagem competitiva dos clientes, necessidade de alterações no escopo ou no time de desenvolvimento no decorrer do projeto, utilização de recursos não previstos, etc. De forma geral, o custo de alterações não planejadas é muito maior do que a alocação de recursos nas fases iniciais do projeto.

Portanto, dada a importância de um planejamento adequado e, por conseguinte, uma estimativa de esforço eficaz, equipes de desenvolvimento ágil utilizam-se de técnicas para mensurar as atividades relacionadas ao desenvolvimento. Dentre algumas técnicas existentes na atualidade, é muito comum que as organizações façam uso do julgamento de especialistas para estimar o esforço por meio do *Planning Poker*. Os Pontos de História são normalmente definidos pela sequência *Fibonacci* e utilizados para mensurar esforço, complexidade e risco de uma atividade durante o *Planning Poker*. Apesar de trazer benefícios, como essa técnica exige debate, consome um tempo da equipe de desenvolvimento e dos especialistas que poderia estar sendo utilizado para atividades que agregam valor ao produto de formas mais práticas e diretas.

Dessa maneira, algoritmos de Aprendizado de Máquina, capazes de identificar padrões, começaram a ser utilizados para automatização da predição de esforço de *software*. Como o aprendizado supervisionado parte da premissa de que existem rótulos - no caso, Pontos de História - anteriormente assinalados ao conjunto de dados de treinamento, um esforço humano deve ter sido despendido na construção da base de dados. Uma vez realizado esse esforço, os algoritmos poderão, estatística e matematicamente, absorver o conhecimento aplicado durante a classificação humana e em seguida automatizar o processo de categorização, dispensando recursos humanos a partir de então.

Foi visto na literatura que a técnica de AM emprega diferentes visões sobre o problema em questão, utilizando bases de dados que empregam diferentes indicadores sobre os projetos de *software*. Indicadores numéricos como custo e quantidade de linhas de código, mais naturais para a linguagem de máquina, são comumente utilizados. Não obstante, algoritmos baseados em indicadores textuais - escritos em linguagem natural - começaram a ser também utilizados, pois as informações de projetos

que são primeiramente disponibilizadas, ainda nos estágios iniciais, são documentos como Histórias de Usuário e Relatórios de Erros.

No universo do Processamento de Linguagem Natural, pré-requisito para sua utilização em algoritmos de Aprendizado de Máquina, este trabalho visou automatizar a estimação de esforço na manutenção de *softwares*, tendo como objetivos específicos atacar diferentes metodologias de extração de características para a classificação textual: BoW, PV-DM e Comitê de PV-DM. Assim, foram executados três *pipelines* compostos por pré-processamento do conjunto de dados, uma das técnicas de extração de características e a classificação dos dados por uma SVM otimizada através de busca randomizada.

Nos experimentos realizados, em todas as abordagens propostas, notou-se uma dificuldade maior de distinção das classes de maior ocorrência na base de dados: $5, 8$ e 13. Apesar da dificuldade em comum, as abordagens analisadas lidaram com essa dificuldade de forma díspar.

A segunda abordagem, a qual utiliza um único PV-DM, foi a que demonstrou melhores resultados, com *f-measure* de aproximadamente 93%. Apesar de alguns problemas inerentes à classificação textual, como a maldição da dimensionalidade, a utilização do método PV-DM, que é uma implementação de Rede Neural e, por conseguinte, tem a capacidade de seleção de atributos, foi capaz de ultrapassar esses problemas. Aliado ao pré-processamento textual realizado, esses fatores conseguiram garantir bons resultados.

A terceira abordagem, aqui proposta, sugere a utilização de um Comitê de PV-DM, na hipótese de que classificadores se especializem em diferentes estruturas do problema. O resultado, apesar de bom, não atingiu as expectativas, provavelmente por falta de recursos na seleção do modelo. Acredita-se, então, que ainda vale a pena um aprofundamento do estudo sobre essa abordagem. Pois, apesar de sua *f-measure* ter sido em torno de 80% - inferior a um único PV-DM -, o comportamento observado pelo modelo apresentou comportamentos similares à abordagem 2 na separação das classes mais difíceis, e comportamentos superiores na separação de classes mais simples (1 e 3).

Devido à falta de recursos computacionais e temporais, a seleção dos modelos dos experimentos foi afetada. À vista disso, uma busca mais extensa pela combinação de hiper-parâmetros dos modelos seria desejável em trabalhos futuros. A correção ortográfica e a remoção de numerais durante o pré-processamento dos textos, assim como a interpretação dos trechos de código, também são atividades a serem consideradas em trabalhos futuros para melhoria da generalização e da predição de esforço. Havendo a disponibilização de bases de dados que contenham o tempo gasto na execução das atividades de correção, também surge uma interessante oportunidade de

aplicar esse atributo nos algoritmos de AM, de maneira a suprir qualquer necessidade de conhecimento humano para, não somente estimar a complexidade, mas também prever o tempo que será, de fato, requisitado para realização das atividades. Por fim, também sugere-se a utilização de bases de dados maiores e mais balanceadas, para garantir o reconhecimento dos padrões contidos nos dados.

Referências

AGGARWAL, C. C.; ZHAI, C. *A Survey of Text Classification Algorithms*. 2. ed. [S.l.]: Springer, 2012. Citado na página 12.

ALLEN, J. F. Natural language understanding. In: *Bejnamin/Cummings series in computer science*. [S.l.: s.n.], 1987. Citado na página 21.

ALMEIDA, E. D. *Algoritmos de Classificação Com a Opção de Rejeição*. Tese (Doutorado) — Faculdade De Engenharia Da Universidade Do Porto, 2010. Citado na página 30.

BAGNO, M. *Gramática Pedagógica do Português Brasileiro*. [S.l.]: Parábola, 2012. Citado na página 25.

BERGSTRA, J.; BENGIO, Y. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, v. 13, n. Feb, p. 281–305, 2012. Citado na página 45.

BISHOP, C. M. *Pattern recognition and machine learning*. [S.l.]: Springer Science+ Business Media, 2006. Citado na página 19.

BOEHM, B. Cocomo constructive cost model originally published in boehm, barry w. *Software Engineering Economics*, 1981. Citado na página 18.

BOURQUE, P.; FAIRLEY, R. E. Swebok, version 3.0: Guide to the software engineering body of knowledge. *IEEE*, 2014. Citado na página 11.

BREIMAN, L. Bagging predictors. *Machine learning*, Springer, v. 24, n. 2, p. 123–140, 1996. Citado na página 31.

CHARETTE, R. N. Why software fails [software failure]. *IEEE spectrum*, IEEE, v. 42, n. 9, p. 42–49, 2005. Citado na página 49.

GALORATH, D. D.; EVANS, M. W. *Software sizing, estimation, and risk management: when performance is measured performance improves*. [S.l.]: Auerbach Publications, 2006. Citado na página 16.

GRENNING, J. Planning poker or how to avoid analysis paralysis while release planning. *Hawthorn Woods: Renaissance Software Consulting*, v. 3, p. 22–23, 2002. Citado na página 19.

GRIMSTAD, S.; JØRGENSEN, M. Inconsistency of expert judgment-based estimates of software development effort. *Journal of Systems and Software*, Elsevier, v. 80, n. 11, p. 1770–1777, 2007. Citado na página 17.

HARRIS, Z. S. Distributional structure. *WORD*, Routledge, v. 10, n. 2-3, p. 146–162, 1954. Citado na página 35.

HILL, J.; THOMAS, L.; ALLEN, D. Experts' estimates of task durations in software development projects. *International Journal of Project Management*, Elsevier, v. 18, n. 1, p. 13–21, 2000. Citado na página 17.

JAIN, R. *Simple Tutorial on SVM and Parameter Tuning in Python and R*. 2017. Acesso em: 20 dez. 2019. Disponível em: <https://www.hackerearth.com/blog/developers/simple-tutorial-svm-parameter-tuning-python-r/>. Citado na página 30.

JENSEN, R. An improved macrolevel software development resource estimation model. In: *1983 5th ISPA Conference*. [S.l.]: Yourdon Press Computing Series, 1983. p. 88–92. Citado na página 18.

JØRGENSEN, M. Practical guidelines for expert-judgment-based software effort estimation. *IEEE software*, IEEE, v. 22, n. 3, p. 57–63, 2005. Citado na página 17.

JUNIOR, N. N. T. *MODELO–E10: UM MODELO PARA ESTIMATIVAS DE ESFORÇO EM MANUTENÇÃO DE SOFTWARE*. Tese (Doutorado) — PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL - FACULDADE DE INFORMÁTICA, 2010. Citado na página 12.

JúNIOR, H. B. da S. *Comparação de Técnicas de Extração de Características para a Estimativa do Esforço no Desenvolvimento de Software a Partir de Documentos em Linguagem Natural*. 2019. Citado 6 vezes nas páginas 11, 15, 18, 19, 39 e 48.

KAPLAN, A.; HAENLEIN, M. Siri, siri, in my hand: Who's the fairest in the land? on the interpretations, illustrations, and implications of artificial intelligence. *Business Horizons*, Elsevier, v. 62, n. 1, p. 15–25, 2019. Citado na página 24.

KOSCIANSKI, A. *Qualidade de Software*: Aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software. 2. ed. [S.l.]: Novatec Editora Ltda., 2007. Citado na página 11.

KUNCHEVA, L. I. *Combining pattern classifiers: methods and algorithms*. [S.l.]: John Wiley & Sons, 2014. Citado na página 40.

LE, Q.; MIKOLOV, T. Distributed representations of sentences and documents. In: *International conference on machine learning*. [S.l.: s.n.], 2014. p. 1188–1196. Citado 3 vezes nas páginas 19, 35 e 37.

LEE, R. C. The success factors of running scrum: A qualitative perspective. *Journal of Software Engineering and Applications*, Citeseer, v. 5, n. 06, p. 367, 2012. Citado na página 18.

LEITãO, D. A. *NIforspec: Uma ferramenta para geração de especificações formais a partir de casos de teste em linguagem natural*. Dissertação (Mestrado) — Universidade Federal de Pernambuco, 2006. Citado 2 vezes nas páginas 21 e 22.

LOCHTER, J. V.; ZANETTI, R. F.; ALMEIDA, T. A. Detecção de opinião em mensagens curtas usando comitê de classificadores e indexação semântica. *Universidade Federal de São Carlos*, 2016. Citado na página 40.

MANNING, C. D.; RAGHAVAN, P.; SCHüTZE, H. *An Introduction to Information Retrieval*. [S.l.]: Cambridge University Press, 2009. Online Edition. Citado 2 vezes nas páginas 19 e 35.

MARTIN, R. C. *Agile Estimating and Planning*. 2. ed. [S.l.]: Prentice Hall, 2005. Citado 2 vezes nas páginas 11 e 42.

MAURYA, A. *Not Just Introduction To Convolutional Neural Networks [Part 1]*. 2018. Acesso em: 20 dez. 2019. Disponível em: <https://becominghuman.ai/not-just-introduction-to-convolutional-neural-networks-part-1-56a36b938592>. Citado na página 28.

MEYER, S. *A Hospedeira*. [S.l.]: Intrínseca, 2009. Citado na página 23.

MINKU, L. L.; HOU, S. Clustering dycom: An online cross-company software effort estimation study. In: ACM. *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*. [S.l.], 2017. p. 12–21. Citado na página 18.

MINKU, L. L.; YAO, X. Ensembles and locality: Insight on improving software effort estimation. *Information and Software Technology*, Elsevier, v. 55, n. 8, p. 1512–1528, 2013. Citado na página 18.

MINKU, L. L.; YAO, X. Software effort estimation as a multiobjective learning problem. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, ACM, v. 22, n. 4, p. 35, 2013. Citado na página 18.

MINKU, L. L.; YAO, X. Which models of the past are relevant to the present? a software effort estimation approach to exploiting useful past models. *Automated Software Engineering*, Springer, v. 24, n. 3, p. 499–542, 2017. Citado na página 18.

NOGUEIRA, P. S. d. S. *Utilizando comitês de classificadores para predição de rendimento escolar*. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Norte, 2015. Citado na página 40.

NUNES, M. G. L. *Processo sintático de coordenação em gramáticas brasileiras contemporâneas do português: sistematização e implicações pedagógicas*. Dissertação (Mestrado) — Universidade Federal de Pernambuco, 2018. Citado na página 25.

OLIVEIRA, A. L. Estimation of software project effort with support vector regression. *Neurocomputing*, Elsevier, v. 69, n. 13-15, p. 1749–1753, 2006. Citado na página 49.

OSMANBEGOVIĆ, E.; SULJIĆ, M.; AGIĆ, H. A review of estimation of software products development costs. *Ekonomski Vjesnik*, JJ Strossmayer University of Osijek, Faculty of Economics, v. 30, n. 1, p. 195, 2017. Citado na página 18.

PATTON, R. *Software testing*. [S.l.]: Pearson Education India, 2006. Citado na página 15.

POLIKAR, R. Ensemble based systems in decision making. *IEEE Circuits and systems magazine*, IEEE, v. 6, n. 3, p. 21–45, 2006. Citado na página 32.

PORRU, S. et al. Estimating story points from issue reports. In: *2016 12th International Conference on Predictive Models and Data Analytics in Software Engineering*. [S.l.]: PROMISE, 2016. p. 01–10. Citado 3 vezes nas páginas 12, 18 e 19.

PRESSMAN, R.; MAXIM, B. *Software Engineering*: A practitioner's approach. 8. ed. [S.l.]: McGraw Hill, 2016. Citado 2 vezes nas páginas 11 e 14.

PROJECT MANAGEMENT INSTITUTE. *Um Guia do Conhecimento de Gerenciamento de Projetos (Guia PMBoK®)*. 6. ed. [S.l.], 2017. Citado 2 vezes nas páginas 11 e 16.

PUTNAM, L.; MYERS, W. Measures for excellence. *Yourdon Press Computing Series*, 1992. Citado na página 18.

RAMOS, A. L.; SANTOS, C. d. Combinando algoritmos de classificação para detecção de intrusão em redes de computadores. *Fortaleza CE*, 2015. Citado na página 40.

RAUDYS, S. J.; JAIN, A. K. Small sample size effects in statistical pattern recognition: Recommendations for practitioners. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, IEEE, n. 3, p. 252–264, 1991. Citado na página 46.

RUBIN, K. S. *Essential Scrum: A practical guide to the most popular Agile process*. [S.l.]: Addison-Wesley, 2012. Citado na página 15.

SAINI, N. *APPLICATION OF EVOLUTIONARY TECHNIQUES FOR SOFTWARE EFFORT ESTIMATION*. Tese (Doutorado) — Delhi Technological University, 2014. Citado na página 17.

SILVA, W. K. N. da; SANTOS, A. de M. Estratégias de construções de comitês de classificadores multirrótulos no aprendizado semissupervisionado multidescrição. *Revista de Informática Teórica e Aplicada*, v. 24, n. 2, p. 71–100, 2017. Citado 2 vezes nas páginas 31 e 40.

SOARES, R. G. F. Effort estimation via text classification and autoencoders. In: *2018 International Joint Conference on Neural Networks*. [S.l.]: Institute of Electrical and Electronics Engineers, 2018. p. 01–08. Citado 5 vezes nas páginas 11, 16, 18, 19 e 38.

SOMMERVILLE, I. *Engenharia de software*. PEARSON BRASIL, 2011. ISBN 9788579361081. Disponível em: <https://books.google.com.br/books?id=H4u5ygAACAAJ>. Citado na página 14.

Só BIOLOGIA. VIRTUOUS TECNOLOGIA DA INFORMAçãO. *Células nervosas*. 2008–2019. Acesso em: 26 nov. 2019. Disponível em: <https://www.sobiologia.com.br/conteudos/FisiologiaAnimal/nervoso2.php>. Citado na página 25.

UEHARA, M. P. de L. *Comparação de Técnicas de Classificação para Predição de Esforço no Desenvolvimento de Software*. 2019. Citado 3 vezes nas páginas 18, 19 e 22.

VINCENT, P. et al. Extracting and composing robust features with denoising autoencoders. In: ACM. *Proceedings of the 25th international conference on Machine learning*. [S.l.], 2008. p. 1096–1103. Citado na página 19.

WEISZFLOG, W. *MICHAELIS: moderno dicionário da língua portuguesa*. Editora Melhoramentos, 2015. Acesso em: 13 out 2019. Disponível em: <www.uol.com.br/michaelis>. Citado 3 vezes nas páginas 11, 20 e 23.

WU, L.; HOI, S. C.; YU, N. Semantics-preserving bag-of-words models and applications. *IEEE Transactions on Image Processing*, IEEE, v. 19, n. 7, p. 1908–1920, 2010. Citado na página 37.

YEATES, D. *Systems Project Management*. 1986. Citado na página 17.

ZIA, Z.; RASHID, A.; ZAMAN, K. uz. Software cost estimation for component-based fourth-generation-language software applications. *IET software*, Institution of Engineering and Technology, v. 5, n. 1, p. 103–110, 2011. Citado na página 17.

ZIAUDDIN, S. K. T.; ZIA, S. An effort estimation model for agile software development. *Advances in Computer Science and its Applications*, v. 2, n. 1, p. 314–324, 2012. Citado na página 17.