Universidade Federal Rural de Pernambuco
Departamento de Estatística e Informática
Bacharelado em Sistemas da Informação

Matheus Pitancó de Lima Uehara

Comparação de técnicas de classificação para predição de esforço no desenvolvimento de software

Recife-PE
2019

MATHEUS PITANCÓ DE LIMA UEHARA

COMPARAÇÃO DE TÉCNICAS DE CLASSIFICAÇÃO PARA PREDIÇÃO DE ESFORÇO NO DESENVOLVIMENTO DE SOFTWARE

Orientador: Rodrigo Gabriel Ferreira Soares

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

Recife-PE

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)

U22c uehara, matheus pitancó de lima
Comparação de técnicas de classificação para predição de esforço no desenvolvimento de software / matheus pitancó de lima uehara. - 2019.
47 f. : il.

Orientador: Rodrigo Gabriel Ferreira Soares.
Inclui referências e apêndice(s).

Trabalho de Conclusão de Curso (Graduação) - Universidade Federal Rural de Pernambuco, Bacharelado em Sistemas da Informação, Recife, 2021.

1. Aprendizado de máquina . 2. Classificação textual. 3. Estimativa de Esforço . 4. Engenharia de Software. I. Soares, Rodrigo Gabriel Ferreira, orient. II. Título

CDD 004

MATHEUS PITANCÓ DE LIMA UEHARA

COMPARAÇÃO DE TÉCNICAS DE CLASSIFICAÇÃO PARA PREDIÇÃO DE ESFORÇO NO DESENVOLVIMENTO DE SOFTWARE

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

Aprovada em: 31 de janeiro de 2019

BANCA EXAMINADORA

Rodrigo Gabriel Ferreira Soares (Orientador)
Departamento de Estatística e Informática
Universidade Federal Rural de Pernambuco

Cícero Garrozi
Departamento de Estatística e Informática
Universidade Federal Rural de Pernambuco

Recife-PE
2019

Agradecimentos

Agradeço primeiramente a Deus, a minha família que sempre me apoiou e tornou minha graduação possível, a minha esposa e minha filha que me deram motivação e força para superar os desafios e conquistar meus objetivos diante das adversidades.

Resumo

A estimação de esforço de atividades é uma etapa fundamental no desenvolvimento de software, ela é de fundamental importância para que o software seja entregue com qualidade no prazo estimado.

Estimativas realizadas de forma isolada das equipes de desenvolvimento tendem a se basear na estimativa de um especialista, essas estimativas são facilmente obtidas porém não refletem fielmente o esforço necessário do responsável pelo desenvolvimento da atividade, enquanto abordagens que envolvem o time de desenvolvimento tendem a ser mais assertivas no entanto demandam mais tempo e mais pessoas envolvidas neste processo de estimativa.

Neste trabalho é apresentado como o aprendizado de máquina pode auxiliar de forma automatizada os times na melhoria de estimativas de esforço diminuindo o tempo necessário para sua realização.

Através dos experimentos foram obtidos resultados que validam a viabilidade da técnica utilizada para extração de características e classificação na estimativa de esforço a partir da descrição textual das atividades. Os resultados dos classificadores variaram de 31% à 33% de F-measure.

Palavras-chave: Aprendizado de máquina, Classificação textual, Estimativa de Esforço, Engenharia de Software.

Abstract

A goal of activity development is critical in software development, and it is critical for the software to be delivered with quality without estimated timeframe.

Estimates were taken from the project houses because they were planned in the one-year forecast, although they were facilitated by not being more stringent in the time of development of the activity, while those involving development time tended to be the more assertive in the semester demand more time and more the whole external forecasting process.

The work was presented as the learning of auxiliary machines in an automated way in the times in which the improvement of movement diminished the time necessary for its accomplishment.

Through the experiments were obtained results that validated the feasibility of the technique used for the extraction of characteristics and classifications in the effort estimate of the textual description of the activities. The values of the classifiers range from 31% to 33% of the F-measure.

Keywords: Machine learning, Textual classification, Estimating effort, Software engineering.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

No ciclo de desenvolvimento de software uma das etapas chave é a estimativa de esforço de atividades, nesse contexto é de fundamental importância uma estimativa assertiva para que o software seja desenvolvido dentro do prazo.

Para realização de estimativas de esforço é necessário que as atividades tenham descrição clara e objetiva além de demandar tempo de muitas pessoas da equipe e que a equipe possua conhecimento de negócio da atividade estimada.

A ausência de algum dos requisitos citados anteriormente pode fazer com que o planejamento e execução do projeto sejam prejudicados, podendo gerar custos adicionais.

Abaixo são descritos cenários gerados a partir de erros de estimativas:

- **Atividades Subestimadas :** Podem gerar maior esforço do time para realização das atividades planejadas, podendo até acarretar em não entrega da atividade.
- **Atividades Superestimadas :** Podem gerar ociosidade na equipe caso não existam atividades para complementar o tempo remanescente do planejamento.

1.1 Justificativa

Estimativa de esforço em software é uma etapa fundamental para o desenvolvimento de um software e pode ser um fator decisivo para sua entrega e implantação no prazo.

Estimativas de esforço demandam disponibilidade de um especialista e dependendo da forma de estimação de esforço também demandam disponibilidade do time.

1.2 Motivação

É de fundamental importância que as estimativas sejam o mais assertivas possível gerando melhor alocação de funcionários, maior previsibilidade de entregas parciais e totais de projetos além de reduzir custos para as organizações.

1.3 Objetivos

Este trabalho tem como objetivo realizar a estimação de esforço de atividades de software de forma automática, confiável e precoce, para isso foi realizado experimentos com os algoritmos de aprendizado de máquina e comparando seus desempenhos na identificação de esforço necessário para realização das atividades.

1.4 Contribuições

As principais contribuições deste trabalho são:

- Utilização de processamento de linguagem natural para realização das classificações das estimativas de esforço em software.
- Comparação do desempenho dos algoritmos de AM na classificação de esforço de relatórios de problemas.

1.5 Organização do Trabalho

Este trabalho está organizado em Catítulos da seguinte forma:

- O capítulo 1 é a introdução deste trabalho, nele é apresentado os problemas atuais da estimativa de software, as motivações e objetivos do trabalho.
- No capítulo 2 são apresentados os trabalhos relacionados ao tema.
- No capítulo 3 é apresentado o referencial teórico utilizado para desenvolvimento do trabalho.
- No capítulo 4 é descrito como foram realizados os experimentos do trabalho.
- No Capítulo 5 é apresentado os resultados dos experimentos e realizado a análise dos resultados.
- No capítulo 6 são apresentadas as conclusões do trabalho.

Capítulo 2

Trabalhos Relacionados

A definição de esforço no desenvolvimento de software vem sendo estudada utilizando diversas abordagens dado a complexidade do problema e grande variação de metodologias de desenvolvimento empregadas na industria.

Este trabalho utilizou uma das bases de dados (TISTUD) do trabalho [3] e um dos métodos de extração de características utilizada no trabalho (PV-DM), também foi utilizado um dos classificadores (SVM) e foram adicionados os classificadores KNN, Rede Neural Artificial, Árvore de Decisão, Adaboost e Bagging. No trabalho citado foi utilizado o título das atividades para sua classificação de esforço de atividades enquanto neste trabalho foi utilizada a descrição das atividades.

Trabalhos como [6] também utilizaram algoritmos de classificação para definição de esforço em projetos de software que utilizam Constructive Cost Model(COCOMO) para definição de esforço, esta abordagem se baseia nos atributos utilizados pelo COCOMO para definição de esforço e os utiliza como entrada para os classificadores, porém esta abordagem ainda necessita de análise dos requisitos para definição de esforço.

Já em [7] o autor propôs a utilização de clustering para melhorar a estimativa de esforço quando os dados de projetos de diversas empresas estão disponíveis, esta abordagem pode reduzir a quantidade de dados necessários para classificações em novas empresas uma vez que dados de outras empresas podem ser incorporados ao treinamento, mas também podem gerar falsas predições pois a estrutura organizacional das empresas é distinta em diversos aspectos.

No trabalho [12] o autor propõe o uso de essembles com diversas configurações para validação de melhorias na estimação de esforço baseado em analogias.

No trabalho [14] é avaliado como obter informações relevantes de projetos anteriores sobre estimação de esforço para predição de esforço em novos softwares de forma automática.

No trabalho [13] é apresentado uma proposta de priorização de histórias de usuário de projetos ágeis utilizando aprendizado de máquina.

A realização de estimativa de esforço utilizando a descrição das atividades para classificação textual podem ter uma importante contribuição para a automação do desenvolvimento de software. Este trabalho pode auxiliar as equipes propondo estimativas assertivas de esforço para realização das atividades

requirindo participação de menos tempo dos membros da equipe no Planning Poker.

Capítulo 3

Referencial Teórico

Neste capítulo é apresentado todo o referencial teórico utilizado para o desenvolvimento do trabalho.

3.1 Estimação de esforço no desenvolvimento de software

O desenvolvimento de software por muito tempo foi realizado utilizando o modelo tradicional de desenvolvimento Figura 3.1 (cascata) onde as etapas eram realizadas uma após o fim das outras, o que fazia com que os projetos fossem entregues e implantados depois de um longo período de tempo, muitas vezes com regras de negócio desatualizadas ou não condizentes com a realidade das empresas.

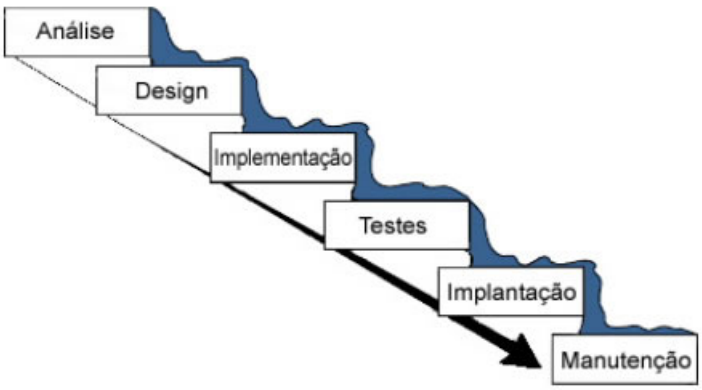

Figura 3.1: Modelo Cascata

Fonte: DevMedia [19]

.

Em contrapartida ao modelo tradicional surgem as metodologias ágeis que tem como principal característica entregas incrementais e curtas sempre validando os requisitos com o cliente.

3.1.1 Scrum

O Scrum é um framework para desenvolvimento e sustentação de produtos complexos. Ele é uma das metodologias ágeis mais utilizadas pelas empresas (TAVARES, 2011), sua definição consiste em[1]:

- **Papeis :** Scrum Master, Product Owner, Equipe de Desenvolvimento
- **Eventos :** Planejamento da Sprint, Daily Scrum, Sprint Review, Sprint Retrospective
- **Artefatos :** Backlog do Produto, Backlog da Sprint, Incremento do Produto
- **Regras :** As regras do scrum estão implicitas nas definições dos seus papeis, eventos e artefatos[2]

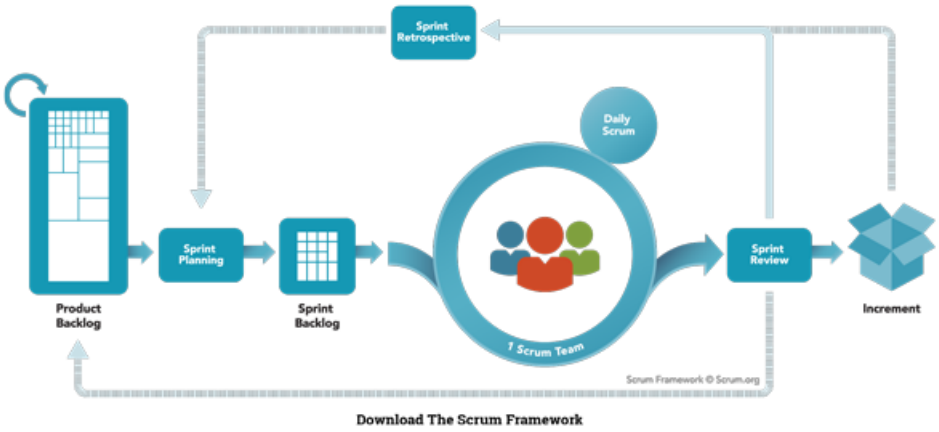

Figura 3.2: Scrum

Fonte: Scrum.org[3]

O Scrum utiliza-se de desenvolvimento (sprints) e entregas (releases) em períodos de tempos fixados afim de obter validação de implementações e melhor alinhamento de expectativas entre necessidades dos clientes e produto entregues Figura 2.2.

Diversos times utilizam como forma estimativa de esforço no scrum, o Planning Poker que é apresentado na Seção 3.1.2 deste trabalho.

3.1.2 Planning Poker

O *planning poker* foi definido e nomeado pela primeira vez por James Grenning, em 2002, e mais tarde popularizado por Mike Cohn, no livro *Agile Estimating and Planning* [17]. Ele busca garantir um consenso entre os membros da equipe na estimativa de esforço das atividades.

Para realização do Planning poker são distribuídas cartas para todos os integrantes da equipe em que cada carta possui um número que serão atribuídos ao tamanho da atividade a ser estimada. Os membros irão cada um atribuir uma carta (tamanho) à atividade até que haja um consenso do tamanho da atividade, em caso de discrepâncias na estimativa os extremos são convidados a defender sua estimativa e o processo é reiniciado com uma nova atribuição.

[1] `https://www.scrumguides.org/scrum-guide.html`
[2] `https://www.scrumguides.org/scrum-guide.html`

3.1.3 Story Points

Story Points podem ser utilizados para medir o esforço para realização de uma atividade de software. Ela é uma unidade de medida abstrata leva que em consideração tempo e complexidade do problema a ser resolvido. Ela normalmente utiliza os números da sequência de Fibonacci em sua representação. Também existem equipes que utilizam a sequência de fibonacci até um determinado tamanho e a partir dele utiliza números que não fazem parte da sequência.

O motivo da utilização de Fibonacci é que os próximos números da sequência são formados pela soma dos dois números anteriores, deste modo não existem atividades com valores muito próximos e o time é forçado a definir um tamanho sem utilizar valores intermediários para as atividades evitando confusão na distinção de atividades e facilitando analogias com atividades estimadas anteriormente devido à padronização dos valores.

Ao fim de cada Sprint é medido quantos Story Points a equipe entregou e assim é atualizada a média de *Story Points* que a equipe consegue produzir por Sprint para que nos próximos planejamentos seja alocadas as atividades corretamente.

3.1.4 Sequência de Fibonacci

A sequência de Fibonacci é uma sequência de números inteiros, na qual cada termo subsequente corresponde à soma dos dois anteriores.

Em termos matemáticos, a sequência é definida recursivamente pela fórmula abaixo:

$F_n = F_{n-1} + F_{n-2}$

Onde os vaores iniciais são:

$F_1 = 1$ e $F_2 = 1$

3.1.5 Problemas em software

Um problema em um software (bug) é um erro ou falha em um programa de computador ou sistema que faz com que ele produza um resultado incorreto ou inesperado [4].

Os erros podem ser obtidos pela equipe de Quality Assurance (QA), por softwares de monitoramento de erros ou pelos usuários do software. A equipe de QA é responsável por validar se o erro realmente existe e criar um relatório de problema.

Os relatórios de problemas são de fundamental importância para que os problemas sejam reproduzidos e corrigidos, garantindo que o software esteja sempre em correto funcionamento.

3.2 Seleção de Características

A seleção de características é responsável por manter apenas as características mais relevantes para a realização da classificação, existem diversas técnicas para realização da seleção de características.

Os principais benefícios da seleção de características são:

[4] `https://en.wikipedia.org/wiki/Software_bug`

- **Redução de Sobre ajuste(Overfitting) :** A seleção de características pode diminuir o sobre ajuste, aperfeiçoando as respostas para dados novos(generalização).
- **Aumento da Acurácia :** A redução de características pouco representativas pode aumentar a acurácia das classificações uma vez que apenas as características mais relevantes para a classificação das instâncias serão utilizadas para a classificação.

3.2.1 F-value

O F-value é um teste estatístico em F-teste. Ele mede a variabilidade entre a média e variância de dois grupos.[1].

F = variância da média do grupo / média das variações dentro do grupo

3.2.2 Qui-quadrado

O x^2 é um teste estatístico que mede a independência de dois eventos, neste trabalho utilizamos ele para medir a independência dos classes(Story Points) e dos termos da descrição do relatório de problema.

A seguir temos a formalização de x^2:

$$\chi^2(t,c) = \sum_{e_t \in \{0,1\}} \sum_{e_c \in \{0,1\}} \frac{(N_{e_t,e_c} - E_{e_t,e_c})^2}{E_{e_t,e_c}},$$

Onde t é definido como o termo e c é a classe, $e_t = 1$ se a descrição contém o termo t e 0 caso contrário. Se a descrição é da classe c, então $e_c = 1$, caso contrário $e_c = 0$. O valor de $N_{t,c}$ é a frequência obtida de cada termo t em c e $E_{t,c}$ é a frequência esperada de cada termo t em c. A estatística de x^2 mede o grau entre os valores esperados e obtidos de $N_{t,c}$ e $E_{t,c}$.

3.2.3 Mutual Information

Mutual Information mede o quanto de informação cada termo contribui para a classificação correta de c, em termos estatísticos ele é descrito como segue:

$$MI(t,c) = \sum_{e_t \in \{0,1\}} \sum_{e_c \in \{0,1\}} p(e_t, e_c) \log_2 \frac{p(e_t, e_c)}{p(e_t)p(e_c)},$$

Onde *p(.)* é a função de probabilidade.

3.3 Extração de Características

3.3.1 Modelo de memória distribuída de vetores de parágrafo (PV-DM)

O modelo Distributed Memory Model of Paragraph Vectors (PV-DM) é uma técnica para extração de características publicada em 2014 por Quoc Lee e Tomas Mikolov. Seu algoritmo de treinamento pressupõe que vetores de palavras contribuem na previsão da próxima palavra em um parágrafo Figura 3.3.

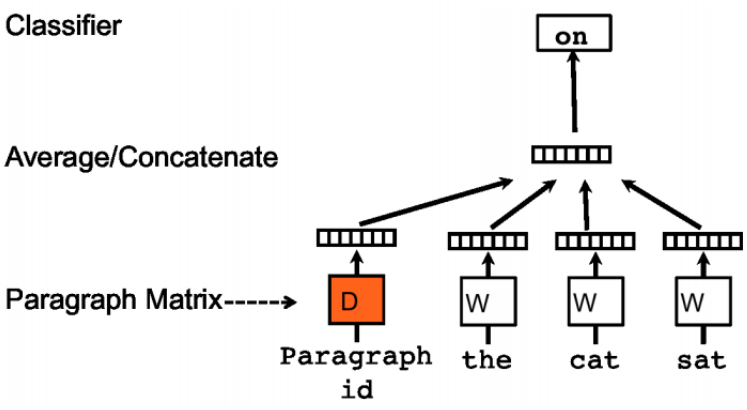

Figura 3.3: Representação da previsão da próxima palavra.

Fonte: [4]

O PV-DM é uma rede neural capaz de transformar representações de textos em vetores de baixa dimensão com um treinamento não supervisionado.

Vetores de palavras codificam a semântica do texto como produto desta previsão. Cada documento é denotado como um vetor com uma coluna de identificação em uma matriz de documentos D e cada termo é representado por um único vetor coluna na matriz dos termos W, os vetores dos termos são atualizados a cada etapa de treinamento e compartilhados por todos os parágrafos.

Os vetores de parágrafo e termo são combinados para prever o próximo termo no contexto de cada texto. O vetor documento representa uma memória que ajuda a indicar o complemento de partes do contexto atual. O comprimento de um contexto é fixo e suas palavras são amostradas a partir de uma janela (window) deslizante conforme ilustrado na Figura 3.4.

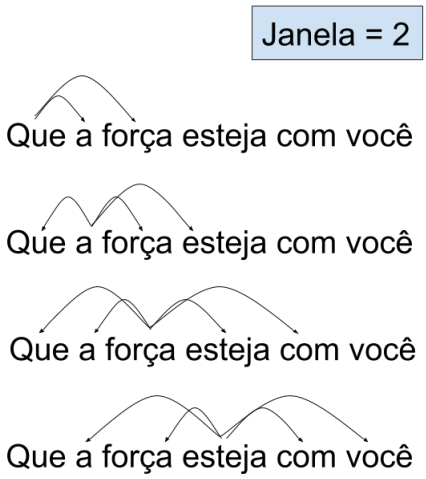

Figura 3.4: Representação da janela

Fonte: O Autor.

O PV-DM emprega um gradiente descentente estocástico via backpropagation e em cada época do treinamento são obtidas amostras a partir de um parágrafo aleatório e calculado a taxa de erro, esta taxa de erro o gradiente utiliza para atualizar os pesos da rede.

3.4 Aprendizado de Máquina Supervisionado

Em aprendizado de máquina para a realização de classificação é dividido a base de dados entre conjunto de treinamento(utilizado para definição do modelo) e testes(utilizado para validação do modelo), após a separação dos dados são obtidas as instâncias pertencentes a cada conjunto e realizados treinamento e validação dos modelos. Abaixo é descrito cada termo utilizado na definição.

- **Instâncias :**As instâncias são todos os registros que compõem a base de dados, eles possuem N atributos e uma classe conforme definidos nos itens seguintes.
- **Atributos :** Os atributos são as características das instâncias, eles são utilizados como paramêtro para aprendizado dos modelos pelos algoritmos de classificação.
- **Classes :** As classes são ”rótulos”que indicam a qual grupo as instâncias pertencem.
- **Vetor de Características -** Vetores de características são representações dos atributos mais relevantes das instâncias obtidas após a realização de extração de características(Seção 4.5.1).

3.4.1 KNN

O K-Next Neighbour(KNN) é classificador utilizado tanto para classificação quanto para regressão. Para realização da classificação ele utiliza como único parâmetro o número de vizinhos K Figura 3.5.

Os 3 passos para que o KNN realize a classificação são:

- **Calcular a distância :** Nesta etapa é calculado a distância da instância a ser classificada para as demais instâncias, para o calculo da distância pode ser utilizado as distâncias Euclidiana, distância Hamming, distância Manhattan e Minkowski.
- **Encontrar os K vizinhos :** Nesta etapa são selecionados os K vizinhos mais próximos da instância a ser classificada.
- **Definido a classe :** Nesta etapa o KNN define a classe da instância classificada baseado na classe predominante dentre os K vizinhos selecionados.

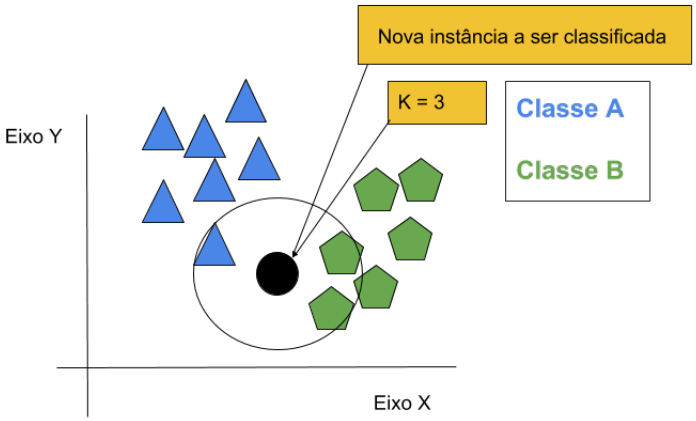

Figura 3.5: Gráfico de Distribuição quantitativa das classes

Fonte: O Autor.

3.4.2 Redes Neurais Artificiais

O modelo de computação da rede neural artificial se assemelha ao modelo biológico do neurônio, este modelo artificial foi desenvolvido por Mc-Culloch e Pitts [5]. Na Figura 3.6 é demonstrado um neurônio biológico e na próxima Seção é descrito o funciona um neurônio artificial.

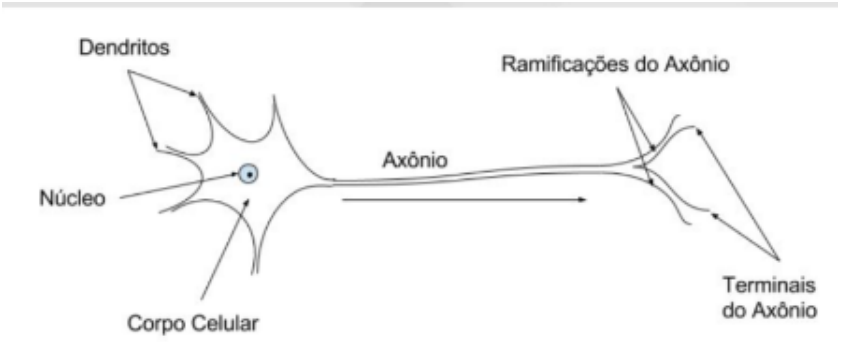

Figura 3.6: Neurônio Biológico

Fonte: O Autor.

3.4.2.1 Perceptron

O modelo perceptron Figura 3.7 possui terminais com n entradas, realizando uma alusão ao neurônio biológico essas entradas seriam os dendritos.

Neurônios artificiais também possui terminal de saída para que o neurônio biológico seja o axônio. São associados pesos aos terminais de entradas do neurônio artificial. Estes pesos são ajustados conforme o aprendizado da rede.

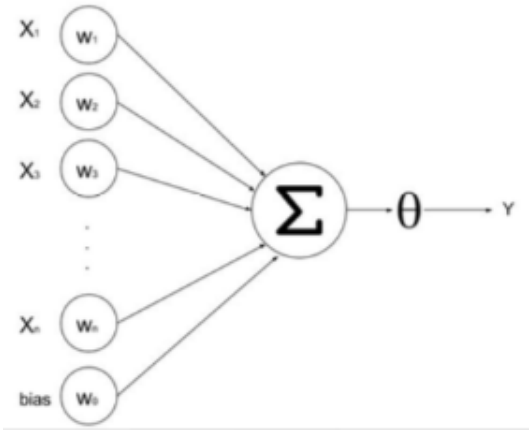

Figura 3.7: Perceptron

Fonte: O Autor.

Em um neurônio biológico, ocorre uma saída (pelo axônio) sempre que a soma do impulso de um neurônio atingiu seu limite. Em um neurônio artificial a saída ocorre através da aplicação de uma função de ativação.

Um dos principais aspectos das redes neurais artificiais é a utilização de uma função de ativação. Ela ativa ou não a saída, dependendo do valor ponderado nas entradas do neurônio.

Existem várias funções de ativação: função de ativação linear, função degrau, função rampa, função sigmóide entre outras [5].

3.4.2.2 Multi Layer Perceptron

Multi Layer Perceptron (MLP) é uma rede neural onde que pode ter várias camadas. A organização da MLP é camada de entrada, camada oculta e camada de saída.

Para o funcionamento da MLP é utilizado a técnica de Feedfoward onde a computação de uma camada é passada para a próxima camada da rede. Caso a próxima camada tenha mais de um neurônio essa saída é replicada para todos os neurônios da próxima camada.

A utilização da técnica Feedforward faz com que este modelo de rede não sofra realimentação em nenhuma de suas camadas, ou seja, cada neurônio se conecta somente com a camada posterior para alimentá-lo e só recebe informação dos neurônios da camada anterior Figura 3.8.

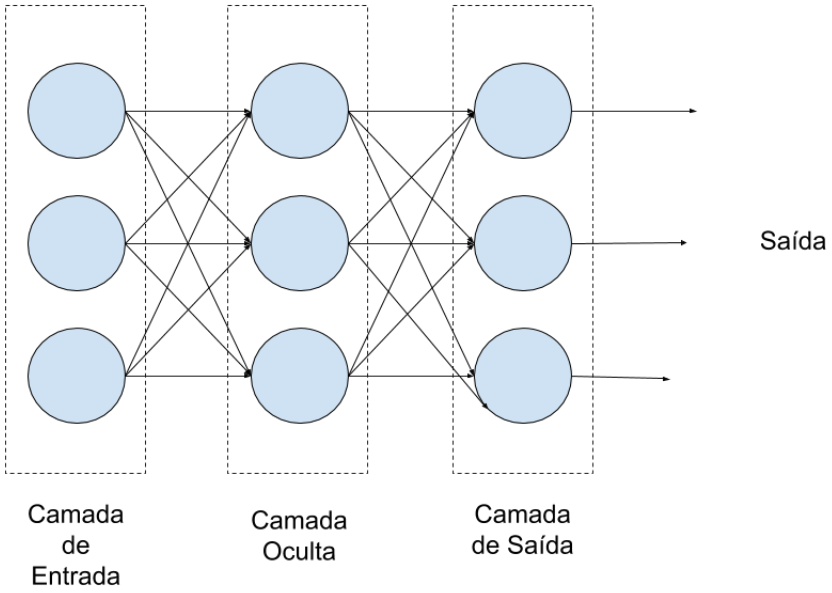

Figura 3.8: Multi layer perceptron que utiliza Feedfoward para alimentação

Fonte: O Autor.

Para realizar o treinamento da rede neural é utilizada uma técnica chamada backpropagation que realiza a atualização de cada peso da rede neural de forma retroativa.

O ajuste dos pesos é feito de acordo com a Regra Delta (BISHOP, 2005) na qual adéqua os pesos da rede através do gradiente descendente.

Abaixo é demonstrado o pseudo-código da atualização dos pesos de um *Multi Layer Perceptron* (MLP).

```
Start with random initial weight s (e.g., in [-.5,.5])
Do {
 For All Patterns p from the training set {
    Calculate Activation
    Error = Target Value_for_Pattern_p - Activation
```

```
    For All Input Weights i {
        DeltaWeight_i = alpha * Error * Input_i
        Weight_i = Weight_i + DeltaWeight_i
    }
 }
}
Until 'Total Error for all patterns < e' Or 'Time-out'
```

Cada neurônio tem sua própria função de ativação, e cada aresta possui um peso específico que vai ser ajustado através da técnica backpropagation.

3.4.3 Árvore de decisão

As Árvores de Decisão são um dos modelos mais práticos e mais usados em inferência indutiva. Este método representa funções como árvores de decisão. Estas árvores são treinadas de acordo com um conjunto de treino (exemplos previamente classificados) e posteriormente, outros exemplos são classificados de acordo com essa mesma árvore.

Para o sucesso de uma árvore de decisão é preciso saber qual o melhor atributo a escolher, para esta tarefa são utilizadas duas métricas:

- **Entropia :** A entropia de um conjunto pode ser definida como sendo o grau de pureza desse conjunto. Este conceito emprestado pela Teoria da Informação define a medida de "falta de informação", mais precisamente o número de bits necessários, em média, para representar a informação em falta, usando codificação ótima.

  Dado um conjunto S, com instâncias pertencentes à classe i, com probabilidade pi, temos:

  $$Entropia(S) = \sum p_i \log_2 p_i$$

  Onde:

  S é o conjunto de exemplo de treino;
  p+ é a porção de exemplos positivos;
  p- é a porção de exemplos negativos;

  A entropia é dada pelo desdobramento da equação: $$Entropia(S) = -p_+ \log_2 p_+ - p_- \log_2 p_-$$

- Ganho: O ganho é definido como a redução na entropia. Ganho(S,A) significa a redução esperada na entropia de S, ordenando pelo atributo A. O ganho é dado pela seguinte equação:

  $$Ganho(S, A) = Entropia(S) - \sum_{v \in values(A)} \frac{|S_v|}{|S|} \cdot Entropia(S_v)$$

3.4.4 Máquinas de Vetores Suporte (SVM)

Máquinas de Vetores Suporte ou Support Vector Machine (SVM) é um algoritmo de aprendizado de máquina(AM) supervisionado que pode ser utilizado tanto para classificação, quanto para regressão (VAPNIK, 1995). Esta técnica originalmente desenvolvida para classificação binária, busca a construção de um hiperplano como superfície de decisão, de forma que a separação entre os exemplos seja máxima.

SVMs lineares são bons para conjunto de dados linearmente separáveis. No entanto, existem muitos casos em que não é possível dividir o conjunto de treinamento linearmente por um hiperplano. Nesses casos, podemos mapear os dados para um espaço de dimensão mais alta, no qual os dados passam a ser linearmente separáveis.

Rótulos ou classe são os fenômenos que desejamos realizar uma previsão. Para SVMs de classificação estes rótulos possuam valores discretos (1,..,n). Nos casos que estes valores sejam contínuos temos o SVM para regressão.

3.4.4.1 Margens Rígidas

O modelo mais simples, que é a base para SVMs mais complexo, são os SVMs lineares com margens rígidas que separam os dados com um hiperplano ótimo, ou seja, de margens máximas. Estes SVMs separam dados linearmente separáveis Figura 3.9.

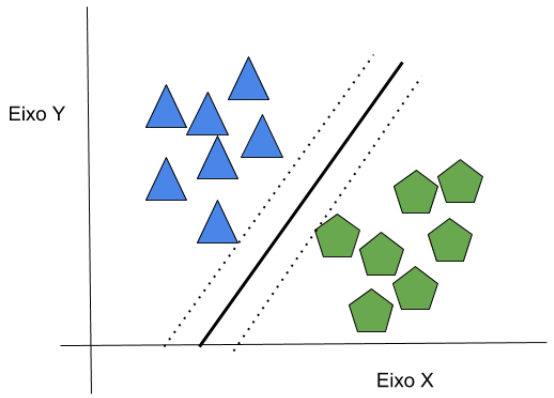

Figura 3.9: Representação do SVM margens rígidas

Fonte: O Autor.

3.4.4.2 Margens Suaves

Como a maioria dos dados reais não são linearmente separáveis, devido a diversos fatores como por exemplo ruídos, o SVM com margem suave introduz uma variável de folga. Esta variável permite que alguns dados possam violar as restrições do SVM com margem rígida Figura 3.10.

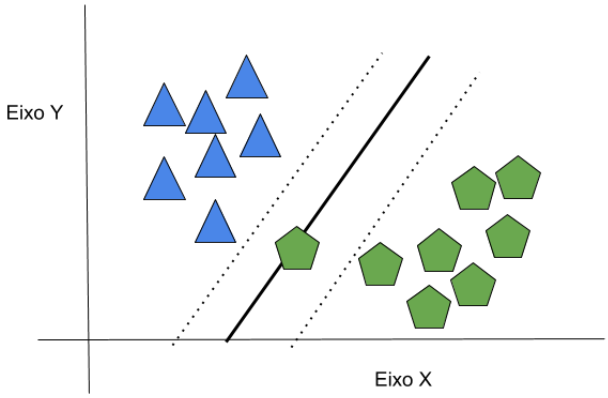

Figura 3.10: Representação do SVM margens suaves

Fonte: O Autor.

3.4.4.3 Máquinas de Vetores Suporte não Lineares

SVM com margens suaves toleram apenas alguns ruídos. Para dados que não podem ser separados por um hiperplano com margem rígida ou suaves, temos o SVM não linear. A técnica adotada aqui é aumentar a dimensionalidade do espaço amostral dos dados, isto é, caso os dados estejam em Rd passamos os dados para o espaço Rx, em que x > d, onde os dados possam ser separados por um hiperplano. Para isso é necessário a aplicação de uma função kernel Soares [2008].

3.4.5 Comitê de Classificadores

Ensemble based systems (EBS), também conhecido como comitê de classificadores, é um método da AM que utiliza a saída de diferentes classificadores base para conseguir uma classificação mais exata. Caso um único classificador não cometer nenhum erro não é necessário a construção de Ensemble.

Caso o classificador escolhido cometa erros, pode-se construir um comitê de classificação com classificadores que não cometam o mesmo erro [10]. desta forma a diversidade na saída dos classificadores base do Ensemble é muito importante. Esta diversidade pode ser realizada de várias formas, como por exemplo utilizando parâmetros diferentes para o mesmo algoritmo, aumentando a quantidade de classificadores base, variando os dados utilizados na construção do classificador, entre outros.

Comitê de classificadores tem mostrado maior desempenho e confiabilidade do que sistemas individuas. Sua dificuldade está em construir um comitê que possui classificadores base com a diversidade necessária. Os principais pontos que deve-se levar em consideração ao combinar os classificadores base são:

- Identificar como realizar a combinação de cada um classificador;
- Criar os classificadores membro;

- Escolher os métodos mais efetivos para o multi-classificador;

3.4.5.1 Bagging

O método baggin Breiman [1999] é bastante utilizado para a construção de comitês, onde os classificadores bases são formados a partir de padrões diferentes. A implementação do bagging é simples, ele foi o primeiro algoritmo construído para implementação de EBS.

No bagging as saídas dos classificadores são combinadas por meio de votos e a classe que obtiver o maior número de votos para uma determinada instância será a resposta.

```
1) model generation
                for i = 1 to T:
                        generate a bootstrap sample D(i) from D
                        let M(i) be result of training A on D(i)
2) prediction for a given test instance x
                for i = 1 to T:
                        let C(i) = output of M(i) on x
                return class that appears most often among C(1)..C(T)
```

A diversidade no bagging é obtida com o uso de diferentes subconjuntos de dados criados aleatoriamente. Cada subconjunto é utilizado para treinar um classificador do mesmo tipo.

3.4.5.2 AdaBoost

O método AdaBoost chama um classificador fraco repetidamente em iterações $t = 1, \ldots, T.$. Para cada chamada a distribuição de pesos D_t. é atualizada para indicar a importância do exemplo no conjunto de dados usado para classificação. A cada iteração os pesos de cada exemplo classificado incorretamente é aumentado (ou alternativamente, os pesos classificados corretamente são decrementados), para que então o novo classificador trabalhe em mais exemplos.

3.5 K-Fold Cross Validation

K-fold cross validation é uma técnica estatística de validação de dados que separa os dados em K grupos, onde 1 grupo é utilizado para teste e K-1 grupos é utilizado para treinamento Figura 3.11.

Com a separação dos dados em vários grupos esta técnica consegue obter o grupo de parâmetros que melhor se adequa aos dados.

Quando K é igual ao número de instâncias K-fold cross validation é igual ao leave one out cross validation.

A técnica utilizada para validação do melhor resultado no Cross Validation é descrita na seção que segue.

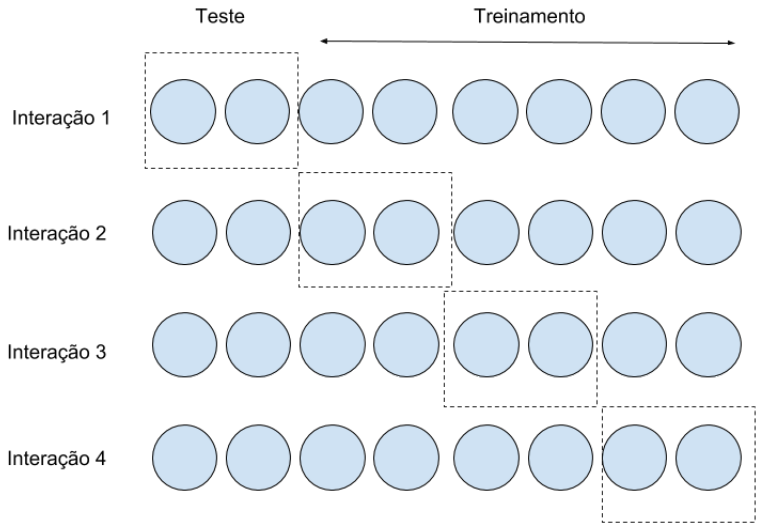

Figura 3.11: K-Fold Cross Validation

Fonte: O Autor

3.5.1 Precision, Recall, F-Measure

- **Recall :** verdadeiro positivos / (falso negativo + verdadeiro positivo);
- **Precision :** verdadeiro positivos / (verdadeiro positivo + falso positivo);
- **F-measure :** 2*((Recall * Precision)/(Recall + Precision)), o F-Measure combina as métricas de Recall e de Precision. O resultado do F-Measure é um indicativo de que, quanto mais próximo de 1, melhor é o algoritmo e resultados mais próximos de 0, demonstram que os algoritmos são piores.

Capítulo 4

Experimentos

4.1 Projeto

O projeto utilizado como fonte de dados para os experimentos foi o Appcelerator studio[1], ele é um ambiente de desenvolvimento integrado (Integrated Development Environment - IDE) para criação de aplicações móveis multi plataforma de nível empresarial utilizando Titanium SDK. Ela foi criada baseado no Eclipse, uma das IDE's mais utilizadas de código aberto.

4.2 Base de dados

Para realização do trabalho foram obtidos os relatórios de problema do projeto open-source Appcelerator Studio (TISTUD) por meio do seu JIRA.

A base de dados a ser utilizada neste trabalho foi escolhida por já ter sido utilizada em trabalhos anteriores[3] para realização de estimativa de esforço a partir de texto (Título das atividades), utilizando aprendizado de máquina e o PV-DM como método de extração de características.

A base de dados obtida possuí 2645 instâncias e 265 atributos. Os registros da base de dados tem data de criação que vão de 01 de março de 2011 à 03 de dezembro de 2017.

4.2.1 Instâncias

As instâncias da base de dados são os relatórios de problemas descritos na Seção 4.4

Abaixo é demonstrado a descrição de uma Instância da base de dados:

```
the javascript parser will report a syntax error in code that functions
properly.

steps to replicate :
1) open titanium studio
```

[1] https://wiki.appcelerator.org/display/guides2/Axway+Appcelerator+Studio+Getting+Started

```
2) open an empty html file and add the content : {CODE BLOCK}
3) note the syntax error on line 4 and add parameter parentheses after
"alertmsg" to correct the error
4) note the editor continues to report a syntax error, save and preview the
file actual results :
the code functions successfully however is marked as syntactically erroneous.

expected results :
the editor should not report errors in valid code.
```

4.2.2 Atributos

A base utilizada nos experimentos possui 265 atributos dos quais apenas 2 foram utilizados:

- **Descrição da Atividade :** Detalhamento do que deve ser realizado para correção de um problema no software
- **Story Point :** Rótulo que informa a qual classe(1,2,3,5,8,13) a instância(Relatório de Problema) pertence

4.3 Pipeline

A Figura 4.1 demonstra o pipeline utilizado para realização do trabalho.

4.4 Relatório de Problemas

Os relatórios de problemas são artefatos criados a partir da ocorrência de problemas no software, eles tem como objetivo documentar o processo de descoberta, reprodução e correção de um problema no software.

Relatórios de problemas possuem as principais informações para que os desenvolvedores possam reproduzir e corrigir os problemas de software, das quais podemos destacar (SOFTWARETESTINGHELP, 2018):

- **Ambiente utilizado nos testes :** Conjunto de hardwares e softwares utilizados para realização dos testes.
- **Versão do software :** Versão do software que foi testado e encontrado o problema.
- **Prioridade :** Prioridade de correção do problema.
- **Gravidade do problema :** Gravidade do problema encontrado.
- **Título do problema**
- **Descrição do Problema**

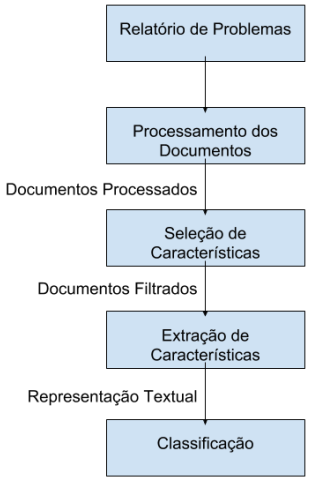

Figura 4.1: Pipeline do Experimento

Fonte: O Autor

O detalhamento de cada etapa do pipeline é realizado nas próximas seções deste trabalho.

- **Tamanho da atividade :** Tamanho estimado da atividade, pode utilizar diversas unidades de medida dependendo da equipe, uma das formas utiliza a sequência de fibonacci conforme Seção 2.3 deste trabalho.

Na descrição do problema podemos ter as seguintes informações adicionais:

- **Passos para reprodução :** Uma explicação detalhada de cada passo necessário para reprodução do problema.
- **Resultados esperados :** Uma explicação detalhada de como o software deve se comportar após realização dos passos descritos.
- **Resultados obtidos :** Uma explicação de como o software está se comportando após realização dos passos descritos.
- **Códigos :** Trechos de código que podem ser responsáveis pelo problema ou Logs do software gerados após realização dos passos descritos.

Na Figura 4.2 é demonstrado um relatório de problema obtido pelo JIRA[2] do projeto AppCelerator Studio[3] nele estão todas as informações necessárias para que a equipe de desenvolvimento realize a correção do problema.

[2] `https://www.atlassian.com/software/jira`
[3] `https://jira.appcelerator.org/browse/TISTUD-8736`

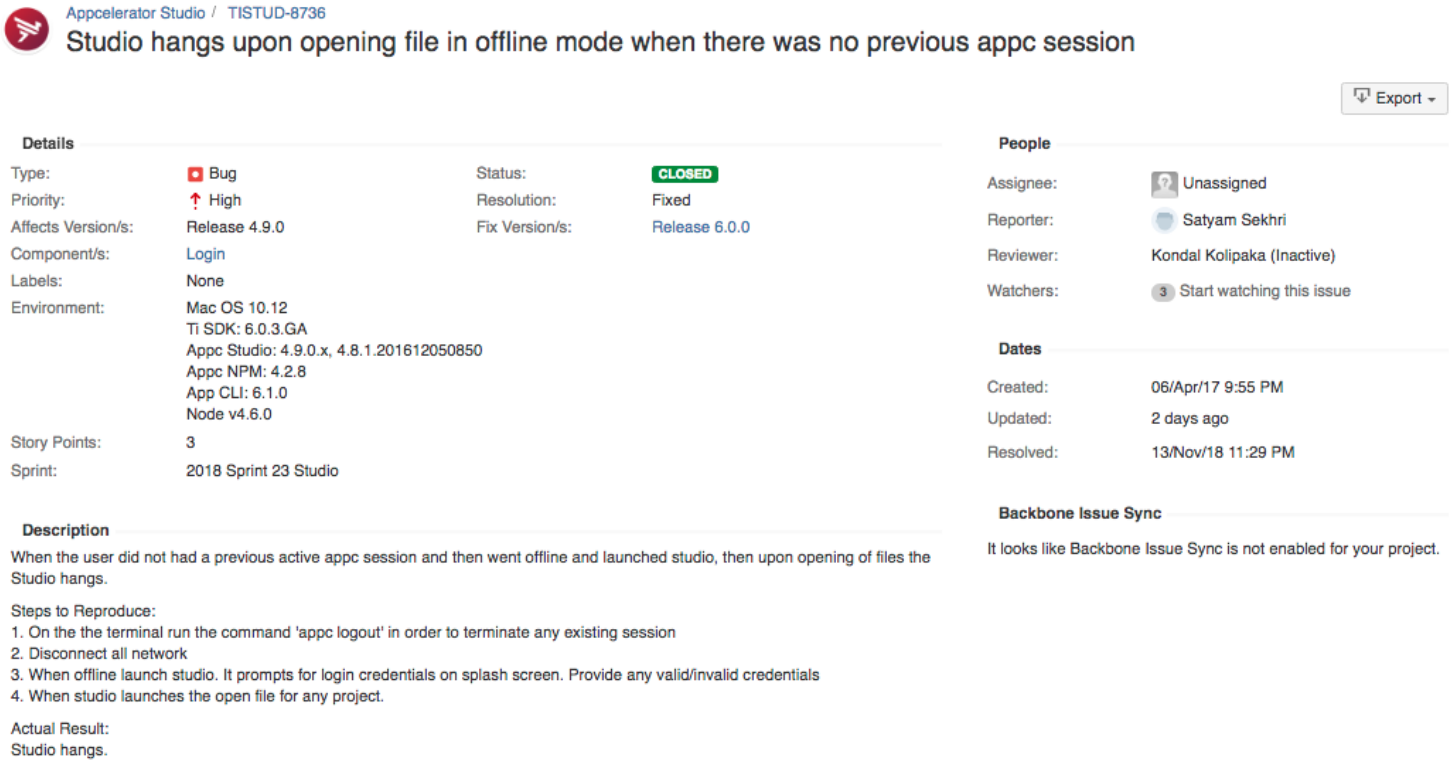

Figura 4.2: Relatório de Problema.

Fonte: Jira Software AppCelerator Studio

4.5 Pré-processamento

No pré-processamento foram realizados os seguintes procedimentos na base.

- **Seleção de Atributos :** Para realização do trabalho foram selecionados 2 atributos para utilização:
  - **Descrição das Atividades :** A descrição das atividades será utilizada como entrada para as próximas etapas do pipeline, é a partir das informações dela que as classificações serão realizadas.
  - **Story Point das Atividades :** O story Point das atividades será utilizado como as classes da classificação, as classes da base de dados são: 1,2,3,5,8,13,21,40
- **Remoção de instâncias :** Foram removidas as classes 21(12 Instâncias) e 40(2 Instâncias) por terem um número muito baixo de Instâncias, além de sua inclusão causar o aumento de 33,3% no número de classes. Também foi realizado remoção de instâncias que não tinham: Descrição ou Story Point preenchidos. Após esta etapa restaram 1426 instâncias na base de dados.
- **Conversão de valores :** Após a remoção dos registros foi realizada a transformação dos textos para minúsculo para que nas próximas etapas as palavras iguais não fossem tratadas como diferentes por conta de diferenças de grafia.

- **Remoção de textos das descrições :** Foi realizada a remoção de stopwords das descrições e também foi realizado a remoção de trechos de código das descrições.
- **Balanceamento :** Após a remoção das instâncias não foi realizado o balanceamento da base de dados pois a realização do balanceamento poderia ser prejudicial no aprendizado, visto que a classe que possuí menos instâncias na base de dados possui apenas 72 registros, isto faria com que a base tivesse seu número total de registros reduzido em aproximadamente 70% do seu tamanho, perdendo muita informação e influenciando negativamente no aprendizado. Por conta disto o método de análise do aprendizado utilizou o precision, recall e f-measure de cada algoritmo como descrito na Seção 5.4.

Após a realização dos itens descritos foram selecionados 1426 instâncias das 2645.

As classes existentes nas instâncias selecionadas são: {1,2,3,5,8,13}. Na Figura 4.3 é demonstrado como está a distribuição das classes nas instâncias.

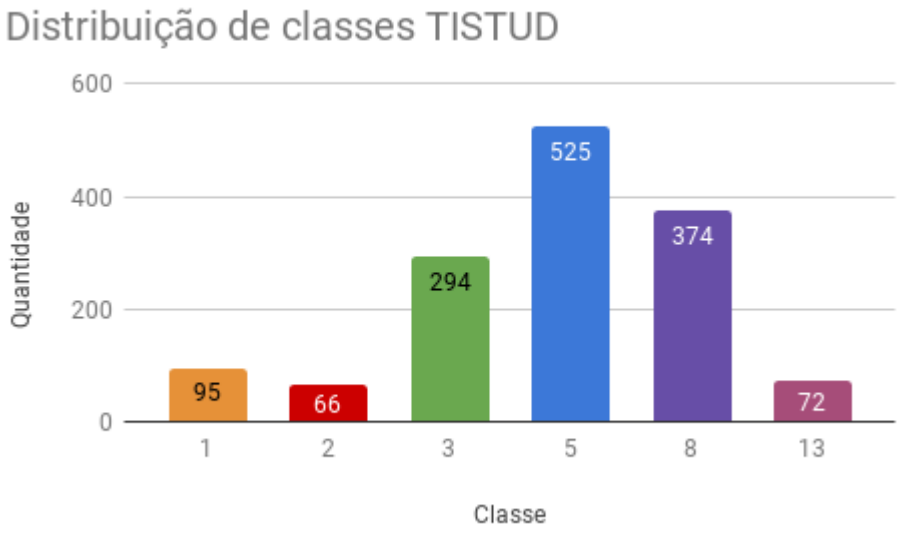

Figura 4.3: Gráfico de Distribuição quantitativa das classes

Fonte: O Autor.

4.6 Seleção de Características

Para realização da seleção de características nesse trabalho foi utilizado a Seleção Univariada. Ela verifica cada característica individualmente para determinar o peso da característica na determinação da variável de resposta. Na seleção de características foram removidas dos textos as características com importância menor do que um percentual.

Neste processo o método utilizado para medir a importância das características foram três estatísticas uni variadas supervisionadas, como sugerido em [8] e [1]: ANOVA F-value (Seção 3.2.1), x^2 (Seção 3.2.2) e Mutual Information (Seção 3.2.3).

Para a realização da seleção de características foi utilizando as classes SelectPercentile[4], chi2[5], mutual_inf_classif[6] e f_classif[7] da biblioteca Scikit learn.

4.7 Extração de Características

Por conta de sua eficácia em vários domínios [4], o PV-DM (Seção 3.3.1) foi a abordagem utilizada no trabalho para a realização da extração de características. O PV-DM realiza a transformação de frase em vetores numéricos que podem ser manipulados pelos algoritmos de classificação.

Para realização da extração de características foi utilizado a implementação Doc2Vec do PV-DM da biblioteca Gensim[8].

Abaixo é demonstrado o vetor de características de uma instância da base de dados obtidos após execução do PV-DM.

```
[-0.09626231, -0.04659135,  0.1049255 , -0.03882215,  0.06901146,
-0.01263005,  0.11218244, -0.00707875, -0.07386144, -0.10547537,
-0.0407869 ,  0.01121413, -0.026311  ,  0.0663536 ,  0.01816273,
-0.03914309, -0.00632125,  0.0914641 ,  0.18590987, -0.02950632,
0.00732425, -0.08641449, -0.02758984, -0.01576762, -0.00823069,
0.01149334, -0.09451603,  0.02581553, -0.06731223, -0.07703158,
-0.0509453 ,  0.00972599, -0.02629647,  0.07856115,  0.02360438,
-0.09250171,  0.02077002,  0.17407258, -0.05156245,  0.00444943,
-0.04909165,  0.03571878, -0.04048252,  0.00740325,  0.03441897,
-0.02424469,  0.09588934,  0.04028442, -0.0887312 , -0.00681308,
-0.04302305,  0.03698575, -0.04243381, -0.09054808,  0.06461225,
-0.02432848, -0.032328  , -0.10393312,  0.02776725, -0.02788496,
0.00452836,  0.0603762 ,  0.06707058,  0.14945312,  0.02750267,
-0.08372117, -0.08196273, -0.05240741, -0.08229747,  0.0017063 ,
0.03340961, -0.07360288, -0.03161754, -0.02999401,  0.02414798,
0.07210385, -0.04531669,  0.0064827 , -0.02077091, -0.04381642,
```

[4] https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectPercentile.html#sklearn.feature_selection.SelectPercentile

[5] https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.chi2.html#sklearn.feature_selection.chi2

[6] https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mutual_info_classif.html#sklearn.feature_selection.mutual_info_classif

[7] https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_classif.html#sklearn.feature_selection.f_classif

[8] https://radimrehurek.com/gensim/index.html

```
−0.01459903, −0.02932599,  0.02615888,  0.0430458 , −0.02596467,
−0.15159875,  0.07158765, −0.08162574, −0.00356239, −0.01634424,
−0.07124571, −0.06021042, −0.03486216, −0.07023866, −0.05004435,
−0.02335993,  0.02403082, −0.02432914, −0.05265241,  0.04378049]
```

Na figura 6.4 temos a distribuição dos dados após a extração de características, para criação da figura foi realizado a redução de dimensionalidade com a utilização da biblioteca TSNE[9]. Pode-se observar que as classes predominantes (3, 5 e 8) encontram-se muito próximas das instâncias das demais classes.

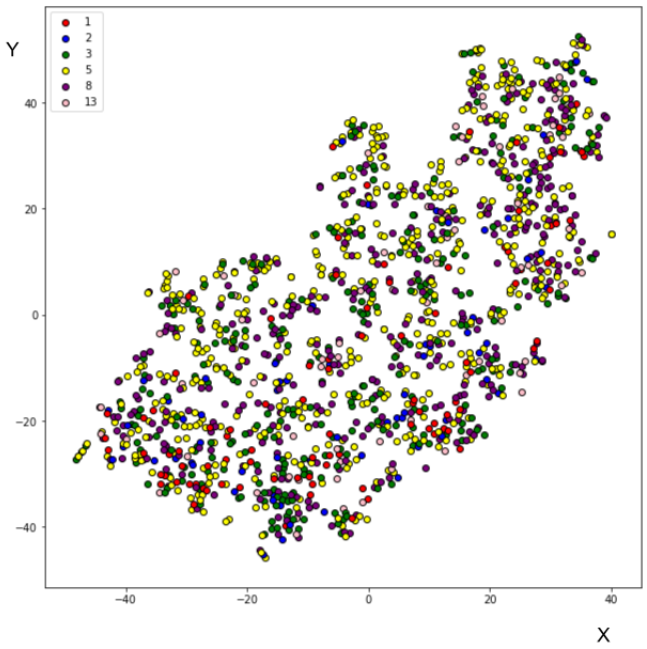

Figura 4.4: Distribuição das classes após extração de características do PV-DM

Fonte: O Autor.

Na figura 6.5 é ilustrado possíveis formações de agrupamentos de instâncias, porém como pode-se observar estes agrupamentos são bastante heterogêneos como as cores das classes demonstram.

Na figura 6.6 é destacado a proximidade das instâncias das classes 1, 3 e 5 o que é ruim para os classificadores. Uma vez que elas tem o vetor de características muito próximos os classificadores podem errar na definição da classe das atividades.

A seguir estão listadas as descrições das atividades destacadas na imagem.

[9] `https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html`

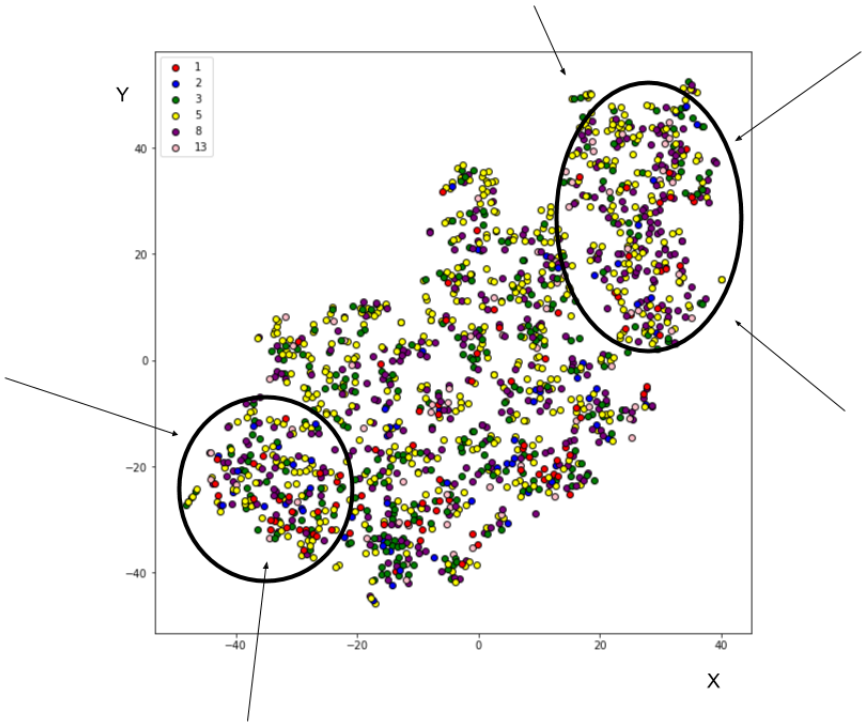

Figura 4.5: Possíveis agrupamentos de dados após extração de características do PV-DM

Fonte: O Autor.

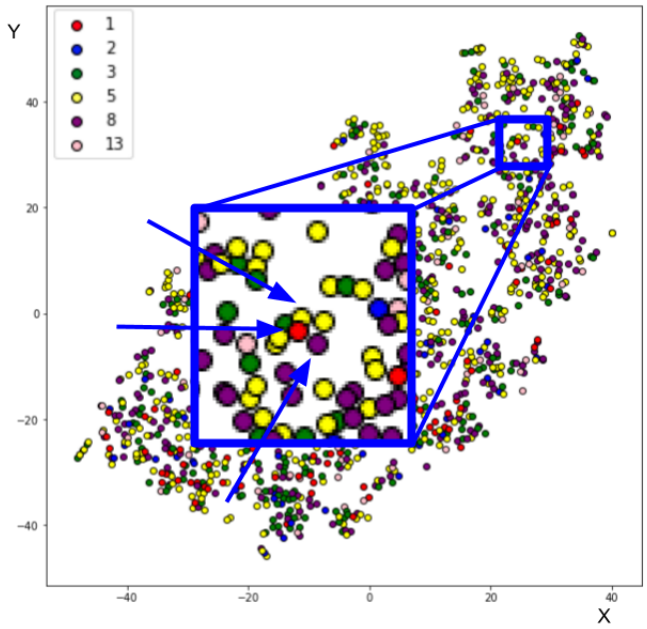

Figura 4.6: Proximidade das instâncias após extração de características do PV-DM

Fonte: O Autor.

- **Atividade da classe 1**

```
steps to reproduce:
1. create a new ios project
2. run as ios device
3. see the console

Actual:
/library/application support/titanium/mobilesdk/osx/1.7.0/iphone
/itunes_sync.scpt: execution error: itunes got an error: can't
continue update. (-1708)

Expected:
no error
```

- **Atividade da classe 3**

```
once we've moved to 64-bit builds in the installer and the ability to
upgrade from 32-bit builds to 64-bit builds, we can remove generating 32-bit
builds on mac entirely.
```

- **Atividade da classe 5**

```
titanium studio, build: 3.2.0.201312081251. mac os

1. in any javascript file, position cursor at beginning of line and
press {{cmd + /}}.

Results: comment not applied to current line.
expected: that.

(related: the associated menu commands are kind of buried under
commands > source > comments. in android dev tools, for comparison,
zthey are the first items under the source menu.)
```

A partir da análise das descrições dos relatórios de problemas da imagem pode-se observar que as atividades eram relacionadas a problemas no sistema operacional Mac OS.

Como o PV-DM utiliza as palavras próximas para a definição do vetor das palavras e a composição dos vetores de palavras para a formação do vetor do documento, é possível que o PV-DM tenha aprendido o contexto do problema e assim gerado vetores semelhantes para as instâncias.

Outro aspecto que deve ser levado em consideração é que as classes (*Story Points*) 1, 2, 3 e 5 são as classes iniciais da sequência de Fibonacci o que significa que as atividades dessas classes são as que tem menor diferença de esforço, o que pode futuramente dificultar a identificação delas pelos classificadores.

4.8 Seleção de modelos

Para realização da seleção de modelos foi utilizado Randomized Search [2], ele faz a combinação dos parâmetros para obtenção do melhor conjunto de parâmetros para cada algoritmo.

Abaixo é listado os valores em que cada parâmetro de cada algoritmo variou.

- **KNN :** Número de vizinhos $\{1, 2, 3, 4...30, 31\}$.
- **Redes Neurais Artificiais :** Taxa de aprendizado $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{1}, 0.3\}$, Número de neurônios escondidos foi de $\{2, 5, 10, 20, 50, 100\}$, Solver variou em $\{'lbfgs', 'sgd', 'adam'\}$ Número de iterações foi de 500.
- **Árvore de Decisão :** minSamplesSplit variou uniformemente em {2..20}, maxDepth variou uniformemente em {1..20}, minSamplesLeaf variou uniformemente em {1..20}, maxLeafNodes variou uniformemente em {1..20} e o Pruned utilizado foi True.
- **SVM :** Kernel $\{rbf, poly, linear, sigmoid\}$, Gamma variou uniformemente de 1^{-10} até 1, Degree variou uniformemente em {1..4}, C variou entre $\{2^{-14}, 2^{-7}, 2^{-3}, 2^{0}, 2^{3}, 2^{7}, 2^{14}\}$.
- **Bagging :** A Árvore de Decisão foi utilizada como classificador e o número de classificadores variou em $\{10, 20, 50, 75, 100, 200\}$.
- **Adaboost :** A Árvore de Decisão foi utilizada como classificador e o número de classificadores variou em $\{10, 20, 50, 75, 100, 200\}$.

Na seção que segue é descrito como foi realizada a validação dos resultados.

4.9 Validação

Para seleção do melhor conjunto de parâmetros dos algoritmos de AM as medidas de precisão (Precision), cobertura (Recall) e F-Measure (Seção 3.5.1) é que foram utilizadas, a validação foi realizada com a técnica Randomized Search [2] e a Validação cruzada (Seção 3.5) com 10 folds.

Capítulo 5

Resultados

Após seleção do melhor conjunto de parâmetros e execução das classificações, foi aplicado o T-test [1] bicaudal em todos os resultados utilizando como parâmetro de comparação o F-measure.

A partir da análise do T-test foram obtidos os resultados apresentados na tabela 5.1.

Algoritmo	*Avg Precision*	*Avg Recall*	*Avg F-Measure*
SVM	0.33±0.03	0.27±0.02	0.33±0.02
Árvore de Decisão	0.29±0.03	0.36±0.03	0.32±0.03
Bagging	0.31±0.04	0.36±0.03	0.32±0.03
Rede Neural	0.32±0.04	0.33±0.05	0.32±0.04
Adaboost	0.30±0.04	0.35±0.03	0.30±0.03
KNN	0.29±0.03	0.36±0.03	0.31±0.02

Tabela 5.1: Tabela Comparativa dos Resultados

As figuras 5.1, 5.2 e 5.3 mostram o Precision, Recall e F-measure dos dados reportados na Tabela 5.1

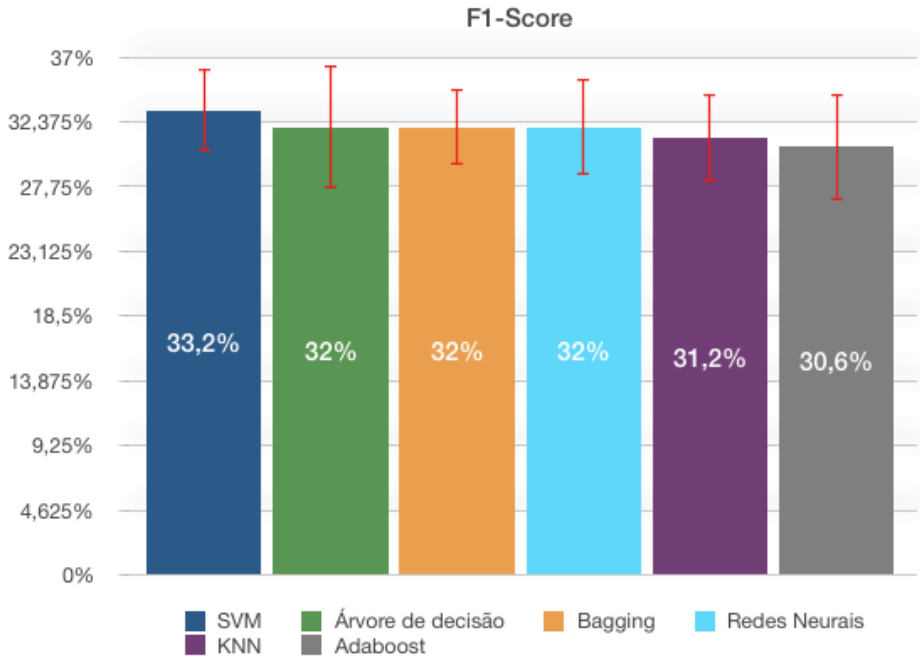

Figura 5.1: F-measure de cada método analisado

Fonte: O Autor.

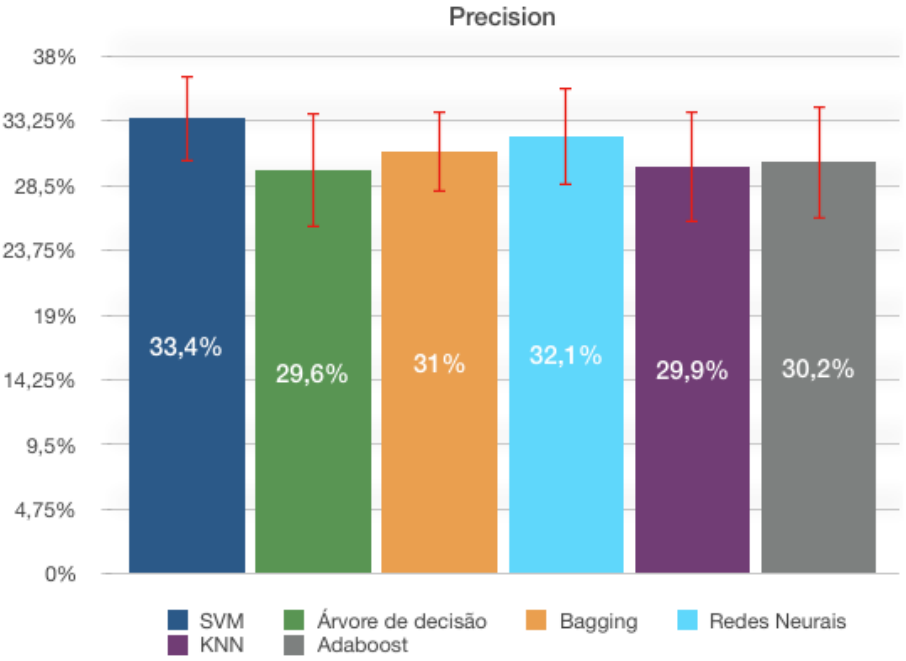

Figura 5.2: Precisão de cada método analisado

Fonte: O Autor.

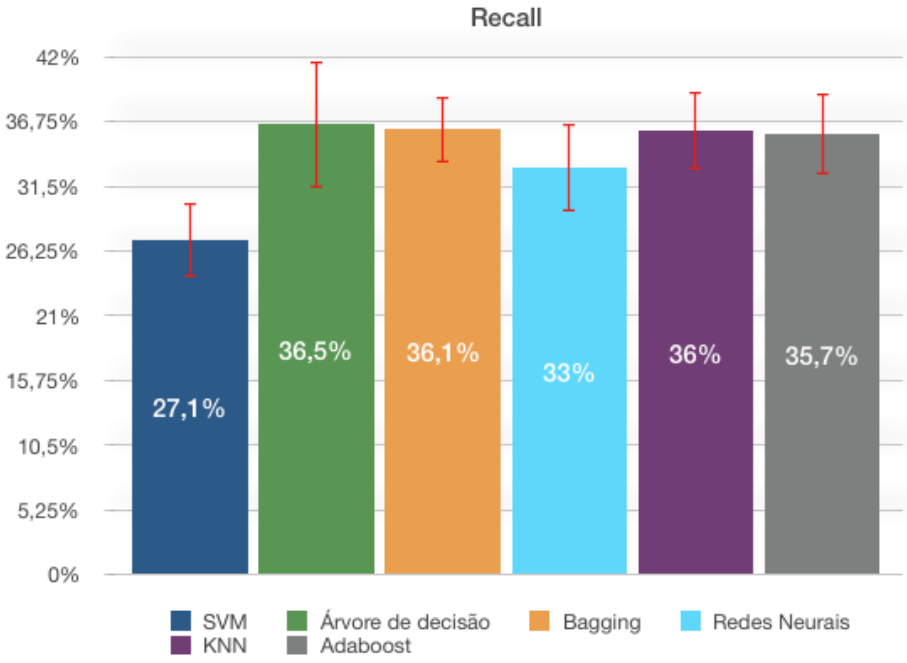

Figura 5.3: Recall de cada método analisado

Fonte: O Autor.

5.1 Análise dos Resultados

Neste trabalho foram realizados experimentos com 6 algoritmos de aprendizado de máquina, KNN, Árvore de decisão, SVM, Redes neurais artificiais(MLP) e o Bagging juntamente com o AdaBoost do paradigma de comitê de classificação, todos aplicados para classificação de dados.

As figuras 5.4, 5.5, 5.6, 5.7, 5.8 e 5.9 das seções que seguem mostram as matrizes de confusão obtidas após classificação com os melhores parâmetros obtidos da validação cruzada, utilizando como instâncias para classificação as mesmas instâncias de treinamento.

5.1.1 Análise do SVM

A partir da análise da matriz de confusão da Figura 5.4 e da tabela de resultados(Tabela 5.1) pode-se observar que a o SVM conseguiu identificar todas as classes da base.

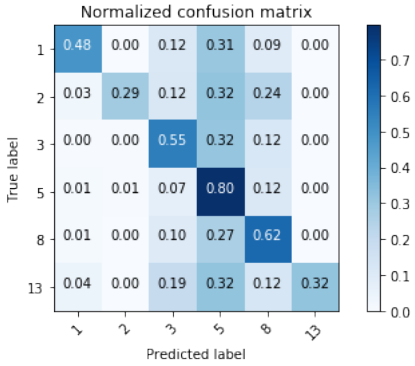

Figura 5.4: Matriz de confusão normalizada do SVM - Instâncias de treinamento

Fonte: O Autor.

5.1.2 Análise da Árvore de Decisão

A partir da análise da matriz de confusão da Figura 5.5 e da tabela de resultados(Tabela 5.1) pode-se observar que a Árvore de Decisão concentrou todas as suas classificações nas classes predominantes 3, 5 e 8.

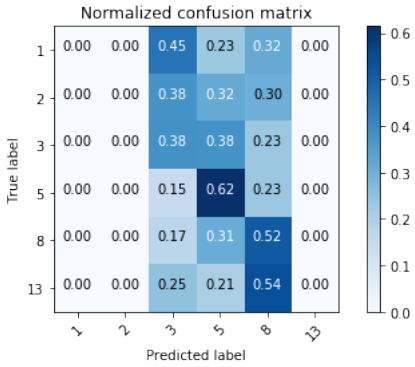

Figura 5.5: Matriz de confusão normalizada da Árvore de Decisão - Instâncias de treinamento

Fonte: O Autor.

Na árvore de decisão fica evidenciado o impacto da base desbalanceada na classificação, onde o algoritmo concentrou todas as classificações em 3 classes e obteve um dos melhores resultados.

5.1.3 Análise do Bagging

A partir da análise da matriz de confusão da Figura 5.6 e da tabela de resultados(Tabela 5.1) pode-se observar que o Bagging teve um overfitting [1] com os dados de treinamento após realização de nova classificação com os mesmos dados.

Embora o Bagging também utilize a composição de Árvores de decisão para sua classificação, estes classificadores funcionam de forma independentes o que pode ter feito com que cada um se tornasse

[1] `https://pt.wikipedia.org/wiki/Sobreajuste`

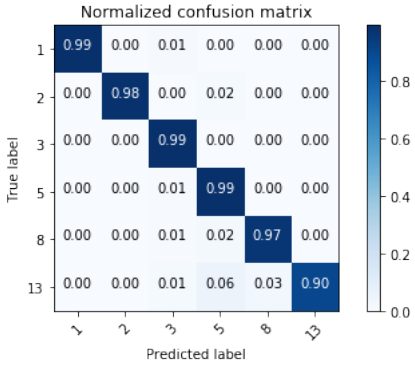

Figura 5.6: Matriz de confusão normalizada do Bagging - Instâncias de treinamento

Fonte: O Autor.

especialista em uma determinada classe e tenha obtido um resultado de quase 100% na nova classificação com os mesmos dados de treinamento.

5.1.4 Análise das Redes Neurais Artificiais

A partir da análise da matriz de confusão da Figura 5.7 e da tabela de resultados(Tabela 5.1) pode-se observar que a rede neural conseguiu identificar todas as classes da base.

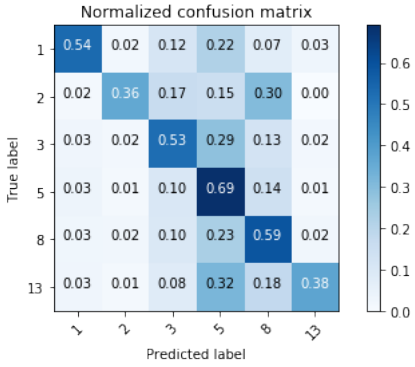

Figura 5.7: Matriz de confusão normalizada da Rede Neural Artificial - Instâncias de treinamento

Fonte: O Autor.

Embora seu f1-score tenha sido o igual a Árvore de Decisão e Bagging a rede neural teve o maior desvio padrão de todos os algoritmos executados, aproximadamente 4% o que fez com que na validação do teste-t o classificador ficasse entre os piores.

5.1.5 Análise do KNN

A partir da análise da matriz de confusão da Figura 5.8 e da tabela de resultados (Tabela 5.1) pode-se observar que o KNN concentrou as classificações nas clases predominantes da base de dados.

Isto deve-se ao fato da base estar desbalanceada e pelo fato do KNN utilizar como base para

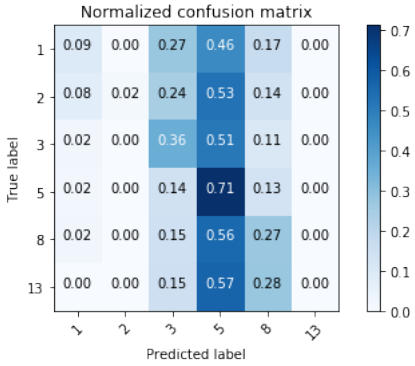

Figura 5.8: Matriz de confusão normalizada do KNN - Instâncias de treinamento

Fonte: O Autor.

classificação os rótulos dos seus vizinhos (Seção 3.4.1). Conforme a figura 4.4 demonstra, as classes predominantes na base de dados estavam dispostas por todas as regiões, o que fez com que o KNN tivesse seu desempenho situado entre os piores classificadores.

5.1.6 Análise do AdaBoost

A partir da análise da matriz de confusão da Figura 5.9 e da tabela de resultados (Tabela 5.1) pode-se observar que o AdaBoost concentrou todas as suas classificações nas classes predominantes 3, 5 e 8.

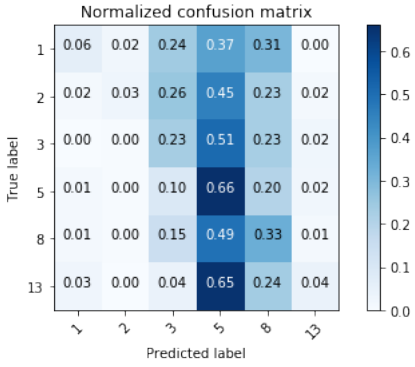

Figura 5.9: Matriz de confusão normalizada do Adaboost - Instâncias de treinamento

Fonte: O Autor.

Podemos atribuir esta concentração na classificação pelo fato do AdaBoost utilizar várias árvores de decisão para composição de sua classificação, o que pode-se observar é que8 houveram classificações de outras classes diferentes de 3, 5, e 8 porém com uma porcentagem muito baixa, isso se deu por conta do Adaboost utilizar aumento dos pesos das instâncias classificadas como errada para os próximos classificadores o que fez com que diferentemente da árvore de decisão o classificador identificasse todas as classes.

Capítulo 6

Conclusões

O objetivo deste trabalho foi avaliar o desempenho de algoritmos de aprendizado de máquina na estimativa de esforço de atividades de software, e buscar dar suporte às equipes nas estimativas de maneira automática, confiável e precoce.

Para isso foram realizados experimentos com bases de dados do software Appcelerator Studio. Foram apresentados trabalhos que utilizavam técnicas de engenharia de software de forma não automatizada para solucionar o problema de estimativa ou abordagens de Aprendizado de máquina baseado em outras características como o título da atividade.

Alguns algorıtimos de AM foram apresentados neste trabalho como proposta para resolver o problema da estimativa de esforço, foram eles: SVM, Redes Neurais(MLP), Bagging, Árvore de Decisão, AdaBoost, KNN.

Como pode ser observado nas Figuras 4.4 e 4.5 após a extração de características atividades de classes distintas estavam formando agrupamentos, o que pode ter dificultado o trabalho de classificação dos algoritmos.

Outro fator que pode ter influenciado na classificação é a falta de concentração das classes predominantes 5 (525 instâncias),8 (374 instâncias),3 (294 instâncias).

Conforme a figura 4.4 as classes predominantes estavam dispostas por todas as regiões, o que pode ter feito com que instâncias das outras classes fossem classificadas como 3, 5 ou 8.

O desbalanceamento da base de dados utilizada pode ter feito com que as classes predominantes tivessem seu aprendizado reforçado.

Analisando o desempenho dos algoritmos de AM propostos neste trabalho, os resultados mostraram com base na métrica F-measure, que os algoritmos SVM, Árvore de Decisão e Bagging obtiveram melhores desempenhos na classificação, enquanto a Rede Neural Artificial o KNN e o Adaboost tiveram os piores desempenhos.

6.1 Trabalhos Futuros

Algumas propostas de trabalhos futuros são:

- Explorar técnicas de aprendizado para bases de dados desbalanceadas, validando se os algoritmos que tiveram mais dificuldade em identificar todas as classes tal como a Árvore de decisão, terão melhora em seu desempenho.
- Realização dos experimentos medindo a variância das classificações realizadas com a classe verdadeira, medindo em quantos graus (classes) os classificadores estão errando, podendo utilizar este grau como parâmetro para a aceitação de uma aplicação prática do trabalho na indústria.
- Realização dos experimentos utilizando um novo atributo criado a partir da junção do título e da descrição das atividades e comparando seus resultados com os trabalhos aplicados anteriormente que utilizaram apenas o título [3] e apenas as descrições das atividades.
- Realização do trabalho em bases de dados maiores e com mais classes.

Referências Bibliográficas

[1] Johnson, R. A., Wichern, D. W. (1992). Applied multivariate statistical analysis. Englewood Cliffs, N.J: Prentice Hall.

[2] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. The Journal of Machine Learning Research, 13, 2012.

[3] Rodrigo G. F. Soares, Effort Estimation via Text Classification And Autoencoders. International Joint Conference on Neural Networks (IJCNN), 2018

[4] Q. Le and T. Mikolov. Distributed representations of sentences and documents. In Proceedings of the 31st International Conference on Machine Learning, volume 32, pages 1188–1196, Beijing, China, June 2014.

[5] BRAGA, A. P.; CARVALHO, A. C. P. L. F.; LUDEMIR, T. B. Redes Neurais Artificiais - Teoria e Aplicações . Ed. 2, p. 225, 2012

[6] Banimustafa, Ahmed. (2018). Predicting Software Effort Estimation Using Machine Learning Techniques. 249-256.10.1109/CSIT.2018.8486222.

[7] MINKU, L.; HOU, S.. "Clustering Dycom: An Online Cross-Company Software Effort Estimation Study", Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering

[8] C. D. Manning, P. Raghavan, and H. Schutze. Introduction to Information Retrieval. Cambridge University Press, 2008.

[9] Rodrigo G. F. Soares. Uso de meta-aprendizado para a seleção e ordenação de algoritmos de agrupamento aplicados a dados de expressão gênica. Master's thesis.

[10] Ludmila I. Kuncheva 2004. Combining Pattern Classifiers: Methods and Algorithms

[11] Eric Bauer Ron Kohavi, Machine Learning 1999. An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants

[12] Mohammad Azzeh, Ali Bou Nassif, Leandro L. Minku, An empirical evaluation of ensemble adjustment methods for analogy-based effort estimation, Journal of Systems and Software,

[13] Alex R. S. Calado, Um sistema de apoio a decisão para priorização e estruturação de histórias de usuários: Um suporte para equipes ágeis

[14] Minku, Leandro L. and Yao, Xin, Which models of the past are relevant to the present? A software effort estimation approach to exploiting useful past models, Automated Software Engineering

[15] BISHOP, C. M. Pattern Recognition and Machine Learning. [S.l.]: Pearson, 2005. ISBN 03-87310-73-8. Citado 12 vezes nas páginas 18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 34 e 36.

[16] VAPNIK, V. The Nature of Statistical Learning Theory. [S.l.]: Pearson, 1995. Citado 3 vezes nas páginas 18, 19 e 21.

[17] Wingman Software | Planning Poker - The Original Paper. https://wingman-sw.com/articles/planning-poker

[18] TAVARES, Gisele. Uso de metodologias ágeis em uma organização baseada em linha de produto. Disponível em: https://www.devmedia.com.br/uso-de-metodologias-ageis-em-uma-organizacao-baseada-em-linha-de-produto-artigo-revista-engenharia-de-software-magazine-38/21662. Acesso em: 26 de janeiro de 2019

[19] CASTRO, Vinicius A. Introdução ao desenvolvimento ágil. Disponível em: https://www.devmedia.com.br/introducao-ao-desenvolvimento-agil/5916. Acesso em 26 de janeiro de 2019

[20] SOFTWARETESTINGHELP, How to Write a Good Bug Report? Tips and Tricks. Disponível em: https://www.softwaretestinghelp.com/how-to-write-good-bug-report/. Acesso em 26 de janeiro de 2019