Classificação Automática de Artefatos De Software: Comparativo Entre Abordagem Manual e Automática De Identificação Atributos

Antônio Maurício Brito Junior
Fortaleza-CE, Brasil, 2021
Email: mauriciobrito@edu.unifor.br

***Resumo*—Neste artigo, fazemos uma análise comparativa entre três abordagens de pré-processamento para a classificação automática de artefatos de software. A primeira abordagem foi a realizada por [1], em que os atributos usados nos algoritmos de aprendizado de máquina foram identificados manualmente por especialistas. As duas abordagens aqui propostas, fazem uso dos vetores TF-IDF *(term frequency–inverse document frequency)* [2] e *words embeddings* e serão discutidas ao longo deste trabalho. Ambas propiciaram melhora de desempenho dos classificadores, comparados com o trabalho de referência.**

Palavras-chave: Software de código aberto, processamento de linguagem natural, classificação de texto, modelo de linguagem, *embedding*, modelo de espaço vetorial, representação distribuída, aprendizado de máquina, artefatos de software.

I. INTRODUÇÃO

A Engenharia de software tem como objetivos melhorar o projeto, o desenvolvimento e a manutenção de produtos de software. Seus principais desafios estão relacionados à crescente demanda dos consumidores, rápida evolução da tecnologia, limitações de tempo para cumprir os prazos estabelecidos, infraestrutura e recursos limitados. Diante de tais desafios, a busca por novas práticas e método tem se mostrado promissor para a adoção de processos que levem a maior produtividade das equipes de desenvolvedores. A abordagem da mineração de dados a partir de repositórios públicos de comunidades de desenvolvedores de software livre, mostra-se uma fonte formidável para estudos das técnicas utilizadas por organizações e profissionais envolvidos na produção de soluções de software. Some-se a esse cenário promissor a grande capacidade que a ciência de dados tem conseguido atingir em gerar novos conhecimentos a partir do uso de técnicas herdadas de Processamento de Linguagem Natural (PLN) com o uso de algoritmos de aprendizado de máquina. Este artigo objetiva contribuir para a adoção de melhores práticas em tarefas de classificação de artefatos de software, utilizando técnicas de PLN e aprendizado de máquina (ML, na sigla em inglês). Ao longo do trabalho, relataremos os experimentos comparativos realizados entre a proposta apresentada por [1] e duas outras abordagens adotadas para o processo de geração de atributos (features): O modelo de espaço vetorial *(Space Vector Model)* usando TF-IDF, e a representação distribuída implementada por meios de vetores densos de palavras pré-treinadas com a classificação realizada por algoritmos convencionais e de redes neurais multicamadas *(Deep Learning)*. Para este estudo, usamos os dados rotulados (oráculo) disponibilizados no trabalho [1], porém o processo de extração de atributos foi realizado automáticamente. Para garantir confiabilidade ao estudo comparativo, adotamos as mesmas métricas do trabalho inicial.

A. Pergunta Motivadora

Como contribuição deste artigo, buscamos responder a seguinte pergunta motivadora: A classificação de artefatos de software terá melhores resultados com a adoção das práticas de PLN? Os resultados comparativos obtidos corroboram a expectativa de que abordagens baseadas em PNL são mais eficazes na tarefa de classificação de artefatos de software do que a abordagem proposta em [1]. Em nossos experimentos, obtivemos 0,98 de F1 Micro Média com o classificador Floresta Aleatória, usando TF-IDF e o vetor de palavras ajustado com o corpus de treino *(Fitted-Word2Vec)*. De um modo geral, obtivemos desempenhos superiores, após realizar o pré-processamento e fazer uso da vetorização com TF-IDF e *word embeddings*.

B. Contribuições Desse Artigo

Neste artigo, apresentamos o estudo comparativo entre três abordagens práticas para a classificação de artefatos de software:

- Abordagem usando um oráculo para gerar os atributos manualmente;
- Extração automática de atributos para uso do modelo de espaço vetorial com TF-IDF;
- Adoção da representação distribuída utilizando vetores de palavras gerados a partir do próprio corpus e vetores de palavras pré-treinados.

C. Trabalhos Relacionados

No estudo de [1], os autores argumentam que a classificação de artefatos de software contribui com informações que possibilitam uma visão ampla dos sistemas de código aberto, como, por exemplo, a rastreabilidade de artefatos como documentação de software pode ser útil para concentrar recursos nisso, economizando tempo e otimizando atividades. Assim, o autor propõe uma abordagem para classificar artefatos de software. Inicialmente, é feita uma análise dos tipos de

artefatos existentes, em seguida, uma heurística é construída para classificar os artefatos que se distinguem pela extensão do arquivo, como imagens e outros. Os arquivos de texto, nos quais seria necessária a leitura do conteúdo, são categorizados em sete classes, sendo estas os alvos dos algoritmos de aprendizado de máquina que classificam os textos. A abordagem proposta para obtenção de recursos é baseada em uma extração manual e os resultados são comparados por quatro algoritmos e um algoritmo de conjunto *(ensemble)* com uma abordagem chamada de classificação por maioria de votos.

No estudo realizado por Vasiliki Efstathiou et al [3], eles discorrem sobre a ausência de vetores de palavras pré-treinados específicos para a área de engenharia de software e como forma de preencher essa lacuna, forneceram um modelo word2vec treinado com mais de 15 GB de dados textuais de postagens do portal Stack Overflow (chamado SO-Word2Vec). Ao longo do artigo, eles ilustram como o modelo elimina a ambigüidade de palavras polissêmicas, interpretando-as dentro de seu contexto de engenharia de software, apresentando exemplos de semânticas refinadas capturadas pelo modelo, que implicam na transferibilidade de resultados para várias tarefas de recuperação de informação direcionadas a engenharia de software , além disso, o estudo busca motivar a utilização do modelo disponibilizado em estudos e obras futuras. Fizemos uso do SO-Word2Vec neste estudo.

No artigo [4], os autores apresentaram os resultados de uma revisão sistemática da literatura na qual foram analisados 1.350 artigos de conferências de engenharia de software dos últimos cinco anos em termos de se eles usaram PNL, qual biblioteca de PNL usaram e como a escolha foi justificada. Os autores descobriram que apenas 33 artigos mencionaram a biblioteca específica utilizada. Além disso, a seleção da biblioteca PNL em particular foi justificada em apenas dois desses artigos. Em seguida, para ajudar os pesquisadores e a indústria a escolher a biblioteca apropriada para analisar artefatos de software escritos em linguagem natural, eles realizaram uma série de experimentos com quatro bibliotecas de PNL diferentes (Google's SyntaxNet, Stanford CoreNLP Suite, NLTK Python Library e spaCy) para artefatos de software disponíveis publicamente de três fontes diferentes (Stack Overflow, arquivos GitHub ReadMe e documentação da API Java). Por meio dos experimentos realizados, os autores concluíram que o melhor desempenho da biblioteca depende da tarefa (tokenização e marcação de classes gramaticais), bem como da fonte utilizada. No geral, SpaCy alcançou a precisão mais promissora.

D. Estrutura do Artigo

Na seção 2, relataremos detalhes da abordagem definida para a realização dos experimentos, com uma descrição das técnicas e ferramentas utilizadas. Na seção 3, apresentamos as métricas de avaliação. Na seção 4, os resultados são relatados e analisados. E, na seção 5, concluímos o estudo e apresentamos oportunidades para trabalhos futuros.

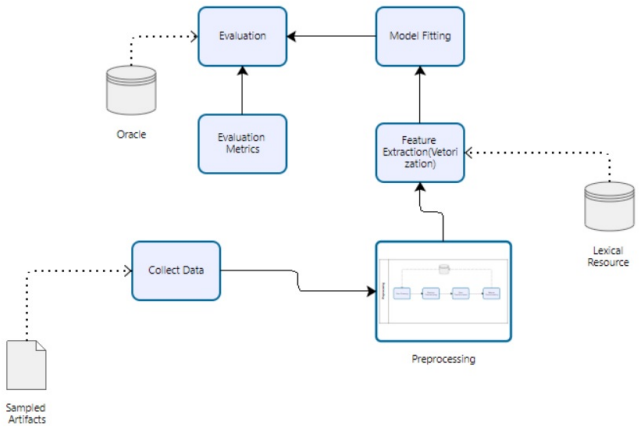

Figura 1: Visão geral do projeto do estudo

II. Projeto do Estudo

Este estudo se propõe a realizar uma análise comparativa da abordagem proposta no artigo seminal [1] e duas outras abordagens para extração automatizada de atributos. Para isso, utilizaremos o modelo de espaço vetorial com TF-IDF e a representação distribuída. Na abordagem com representação distribuída, realizaremos experimentos com vetores de palavras *(embeddings)* treinado no corpus de dados de treinamento com Gensim [5], um vetore pré-treinado de domínio específico e dois vetores genéricos. Em ambas as abordagens, usaremos vários algoritmos convencionais de aprendizado de máquina supervisionado e redes neurais artificiais. A Figura 1 apresenta uma visão geral das etapas realizadas neste estudo.

Nossa motivação inicial nos levou às seguintes perguntas investigativas:

PI1 - Qual classificador obterá melhor desempenho e em qual vetorização?

PI2 - Na representação distribuída, um vetor de palavra de domínio específico terá melhor desempenho do que um vetor de palavras genérico?

Para responder à pergunta PI1, o conjunto de dados foi lido com as categorias de artefatos de software rotulados (conhecidos), realizada a etapa de pré-processamento para a extração dos atributos. Em seguida, realizamos a vetorização (em dois momentos diferentes para fazer uso do modelo TF-IDF e dos vetores de palavras). Para as comparações, implementamos os mesmos algoritmos usados por [1]. Adicionalmente, realizamos experimentos com novos algoritmos utilizando redes neurais. Os classificadores usados foram aqueles disponíveis nas bibliotecas *Scikit-Learn* e *Keras*. Em todos os experimentos, usamos validação cruzada de 10 vezes *(10 folds Cross Validation)*.

Para responder a PI2, usamos quatro vetores diferentes:

- SO-Word2Vec - Um vetor semelhante ao word2vec treinado com mais de 15 GB de dados textuais de postagens do Stack Overflow construído e disponibilizado por [3];
- Um vetor de palavras treinado com o corpus dos dados de treino disponíveis e utilizando a biblioteca gensim [5];

- GloVe 42B.300d [6]'- É um vetor feito a partir de um corpus público da web *(Common Crawl)*, treinado a partir de 42 bilhões de tokens, contendo 1,9 milhões de vocábulos, em letra minúsculas no idioma inglês, com 300 dimensões e download de 1,75 GB;
- Bert - É um vetor baseado em *Transformers* [7]. Neste estudo, utilizamos o Bert Básico que possui 12 camadas e 110 milhões de parâmetros, em letras minúsculas no idioma inglês.

Nas subseções a seguir, apresentaremos os conceitos e técnicas que utilizamos.

A. Classificação de Texto

Linguagem são dados não estruturados que foram produzidos por pessoas para serem compreendidos por outras [8]. O grande maioria de artefatos de software possui essas mesmas características. Eles foram produzidos como parte de manuais e documentos que orientam o processo de desenvolvimento ou servem como comprovação das etapas executadas. A classificação é uma das atividades mais fundamentais na análise de textos. Basicamente, consiste em identificar padrões de dados definidos e rotulados e determinar a quais categorias esses padrões estão relacionados. Como o objetivo é fornecido antecipadamente, a classificação é considerada aprendizado de máquina supervisionado, pois um modelo pode ser treinado para minimizar o erro entre as categorias previstas e reais nos dados de treinamento [8].

B. Características dos Dados Amostrais

Os dados utilizados foram aqueles disponibilizados pelo trabalho inicial [1], no qual os artefatos de software são classificados em 7 categorias (descritas a seguir). A figura 2 mostra a distribuição de frequência das amostras rotuladas. Observa-se que a amostra é pequena, composta por 208 documentos, e desbalanceadas. Esses dois fatores frequentemente contribuem para o desempenho ruim na maioria dos classificadores [5], [9], [10]. Para mitigar esses problemas, várias ações foram tomadas, como métodos de nível de dados para lidar com o desbalanceio das categorias [11], abordagens de validação cruzada aninhadas *(nest cross-validation)* e separação dos dados em um conjunto de treinamento e outro de teste *(train-test-split)* [12], para obter resultados imparciais e melhorar o desempenho dos classificadores, independentemente do tamanho da amostra utilizada.

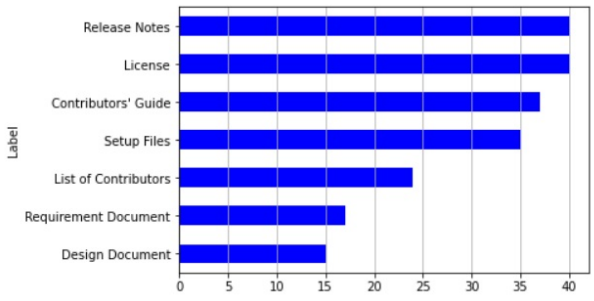

Figura 2: Distribuição por categorias

C. Categorias

Abaixo está uma descrição detalhada de cada categoria de artefato de software definida e utilizada neste artigo:
1 - Documento de Projeto *(Design Document)*: Esses arquivos contêm informações sobre o projeto, como normas e decisões definidas. Também é comum conter dados sobre a estrutura e compatibilidade entre versões.
2 - Documento de Requisito *(Requirement Document)*: Arquivos contendo requisitos não funcionais, casos de uso e outras especificações formais aplicadas no contexto da engenharia de software.
3 - Licença *(Licence)*: A licença contém informações de direitos autorais, descrevendo os tipos de licenças sob os quais o projeto opera.
4 - Lista de Colaboradores *(List of Contributors)*: Contém informações gerais sobre os colaboradores e mantenedores do projeto, neste documento é comum conter nomes, contatos e funções de cada colaborador.
5 - Notas de versão *(Release Notes)*: geralmente, este tipo de arquivo é compartilhado com o usuário final, contém notas sobre a versão e geralmente descreve mudanças específicas para o projeto, bem como correções de bugs e melhorias.
6 - Guia do Colaborador *(Contributor's Guide)*: Guia do Colaborador: Este tipo de arquivo traz informações direcionadas aos colaboradores do projeto.
7 - Arquivos de instalação *(Setup Files)*: Contém todos os artefatos que especificam as configurações do projeto.

D. Pré-processamento

O principal objetivo do pré-processamento é obter os principais atributos do documento de texto que possuam a maior relevância com a categoria definida para aquele documento [13]. Nos dados brutos, erros gráficos são comuns, portanto, seguindo as boas práticas de PNL, começamos com a tokenização dos textos e remoção de pontuação ortográfica, também fizemos remoção de palavras comuns *(stop words)* sem muita relevância para adicionar significado ao documento, depois a transformação dos dados (transformação em letras minúsculas) e construção de n-gramas. O objetivo dessa etapa é eliminar todos os atributos não informativos, que não contribuem para a tarefa de classificação de texto. Isso não só produz uma melhor precisão de generalização, mas também reduz os problemas de sobreajuste *(overfitting)* [14].

E. Classes Desbalanceadas

Conforme mencionado, nosso conjunto de dados tem poucas amostras e classes desbalanceadas. Esse tipo de situação ocorre em domínios do mundo real, onde sistemas de tomada de decisão visam detectar os casos raros, mas de suma importância. Problemas com conjuntos de dados desbalanceados têm recebido muita atenção nos últimos anos e várias técnicas estão surgindo para mitigá-los. Em nosso caso, usamos a técnica de sobreamostragem aleatória *(random oversampling)* para aumentar as amostras de classes minoritárias. A sobreamostragem aleatória é um método que visa equilibrar a distribuição de classes através da replicação aleatória de exemplos de

classes minoritárias [11]. As classes de Documento de Projeto, Documento de Requisito, Lista de Colaboradores, Arquivos de Instalação e Guia do Colaborador foram aumentados com a sobreamostragem aleatória.

F. Seleção de Atributos

A seleção de atributos é o processo no qual os atributos que mais contribuem para a previsão em que você está interessado são selecionados automática ou manualmente. Realizamos um teste de qui-quadrado para determinar se um atributo e o alvo *(target)* são independentes e mantivemos apenas os atributos com o valor de p maior ou igual a 0,9 no teste de qui-quadrado [15].

G. Extração de Atributos

Para realizar o aprendizado de máquina em texto, precisamos transformar nossos documentos em representações vetoriais para que possamos aplicar o aprendizado de máquina. Este processo é chamado de extração de atributos ou mais simplesmente, vetorização, e é um primeiro passo essencial para a análise do conhecimento da linguagem [8]. A forma mais comum de transformar texto é usar o chamado modelo de espaço vetorial (VSM, na sigla em inglês), onde os documentos são modelados como elementos em um espaço vetorial [16], [17]. Neste estudo, usaremos uma implementação VSM amplamente utilizada e eficaz, o modelo *Term Frequency - Inverse Document Frequency* (TF-IDF), onde uma noção básica de relacionamento semântico é incorporada ao vetor gerado. A ideia central desse modelo é que os termos mais raros de um documento sejam codificados com maior relevância estatística [2].

H. Representação Distribuída

A representação distribuída cria um vetor multidimensional que permite armazenar informações sobre a semelhança ou relação semântica entre as palavras em um corpus [18]. Esses vetores de palavras também têm as vantagens de terem menos dimensões do que, por exemplo, vetores TF-IDF. Neste estudo, usamos Word2Vec, SO-Wor2Vec, GloVe e Bert.

No caso do Bert, embora bem adequado para lidar com sentenças relativamente curtas, sua tecnologia baseada em transformadores, faz com que tenha limitações em sua aplicabilidade na classificação de sentenças longas, ou seja, são capazes de processar apenas um contexto limitado de símbolos como sua entrada [19]. Neste trabalho, os documentos que usamos como amostra para classificação possuem, em média, mais de 1.100 palavras, por isso, adotamos uma abordagem baseada no trabalho de [19], no qual dividimos as *strings* de texto de entrada em segmentos menores de comprimento fixo, com uma sobreposição para dar continuidade aos significados das frases originais. Cada nova frase é atribuída à mesma classe da frase original e um novo conjunto de dados de treinamento é criado. Após esta etapa, enviamos para o Bert obter um *embdeding* para cada palavra da frase.

I. Algoritmos de Aprendizado de Máquina

Implementamos os mesmos algoritmos usados no trabalho feito por [1], mas usando extração automática de atributos com TF-IDF. Adicionalmente, realizamos os experimentos utilizando um classificador com arquitetura de rede neural recorrente bidirecional (BRNN) [20]. Abaixo está um resumo dos algoritmos usados.

1 - Florestas Aleatórias *(Random forests)* Florestas aleatórias é um algoritmo de aprendizado supervisionado criado por Leo Breiman [21]. É um algoritmo muito flexível e fácil de usar. Uma floresta é feita de árvores. Diz-se que quanto mais árvores você tem, mais robusta é a floresta, isso também reduz a possibilidade de sobreajuste *(overfitting)*. Ele funciona da seguinte maneira: o algoritmo seleciona amostras aleatórias de um determinado conjunto de dados e, em seguida, constrói uma árvore de decisão para cada amostra e obtém um resultado de previsão de cada árvore de decisão. Faz uma votação para cada resultado previsto e a previsão que obtiver mais votos será o resultado final.

2 - Multinomial Näıve Bayes O classificador multinomial Näive Bayes é baseado na aplicação do teorema de Bayes [22], onde se assume que as características do conjunto de dados são mutuamente independentes. A ocorrência de um atributo não afeta a probabilidade de que o outro atributo ocorra. Para tamanhos de amostra pequenos, Naïve Bayes pode superar as alternativas mais poderosas.

3 - Ärvore de Decisão *(Decision tree)* A árvore de decisão é um algoritmo de aprendizado de máquina supervisionado onde os dados são continuamente divididos de acordo com um determinado parâmetro para exemplos de classificação.

4 - Aprendizado de Conjunto ou Comitê *(Ensemble Learning)* Ensemble Learning é uma técnica aplicada para combinar classificadores individuais a fim de obter melhor eficiência preditiva geral [23]. Para tanto, foi utilizado o algoritmo de votação majoritária, que representa uma abordagem de eleição coletiva que considera os votos de cada classificador para o rótulo de uma instância e utiliza a classificação mais votada.

5 - Máquinas de Vetores de Suporte *(Support Vector Machine)* O algoritmo de máquinas de vetores de suporte (SVM, na sigla em inglês) é um classificador binário, que pode ser estendido para várias estratégias de classe, que podem ser aplicadas com sucesso a diferentes tipos de problemas e complexidade [24]. Neste trabalho, usamos a implementação disponível em Scikit-Learn [22].

6 - Redes Neurais Recorrentes Bidirecionais *(Bidirectional Recurrent Neural Networks)* redes neurais recorrentes (RNN) se tornaram um dos modelos mais usados em tarefas de processamento de linguagem natural. Essa popularidade se deve ao fato de sua arquitetura ser bastante adequada para fornecer textos de tamanho variável. Eles também demonstraram grande capacidade de aprendizado. Para implementar o RNN, usamos um LSTM bidirecional da biblioteca Keras, para aprender as propriedades do texto, porque o significado de uma palavra não está apenas relacionado ao conteúdo do texto anterior a ela, mas também ao conteúdo do texto posterior a ela [25]

[20]. Neste estudo, a arquitetura do modelo consiste em uma camada de embedding que leva sequências como entrada e vetores de palavras como pesos, duas camadas LSTM bidirecionais para modelar a ordem das palavras em uma sequência em ambas as direções e para entender quais partes de um texto longo são realmente relevante. E, finalmente, uma camadas densa que irá prever a probabilidade de cada categoria de artefato de software. Entre as camadas densas, colocamos uma camada Dropout para evitar sobreajuste. Os parâmetrosda rede foram ajustados para obter o melhor desempenho para cada vetor de palavras utilizado.

III. AVALIAÇÃO

Nesta seção, descreveremos as técnicas e métricas usadas para a avaliação. Como nosso objetivo é realizar um estudo comparativo, as mesmas métricas utilizadas em [1], também foram usadas aqui.

A. Validação Cruzada (Cross Validation)

A validação cruzada (CV, da sigla em inglês) é um procedimento de reamostragem usado para avaliar modelos de aprendizado de máquina, amplamente adotado quando a amostra de dados é limitada. K-Fold é uma abordagem de CV comum usada principalmente para estimar a capacidade de um modelo de aprendizado de máquina de fazer estimativas a partir de pequenas amostras e prever dados não vistos durante a fase de treinamento. O método consiste, em primeiro lugar, um modelo bem definido é desenvolvido normalizando dados, selecionando atributos, parâmetros de ajuste e/ou executando outras etapas de desenvolvimento. Em seguida, uma parte dos dados é separada para validação, deixando o resto para treinar o modelo e prever as classes nos dados de validação deixados de fora. Este processo é repetido várias vezes, deixando de fora uma parte diferente dos dados para validação até que todos os dados sejam usados. O desempenho do modelo é então calculado como uma média dos desempenhos de classificação, em cada uma das vezes *(folds)* de validação [12]. Usamos uma validação cruzada de 10 vezes.

B. Precisão

A precisão denota a proporção de casos positivos previstos que são corretamente positivos reais [26] e expressa pela fórmula: Precisão = TP / (TP + FP), onde TP e FP são o número de verdadeiros (true) e falsos (false) positivos, respectivamente.

C. Recall

Recall ou sensibilidade (como é chamado em psicologia) é a proporção de casos reais positivos que são corretamente preditos positivos [26]. Lembre-se = TP / (TP + FN), onde TP e FN são o número de verdadeiros positivos e falsos negativos, respectivamente.

D. Métrica F (F-Measure)

A métrica F1 combina precisão e recall para trazer um número exclusivo que indica a qualidade geral do seu modelo. A F1 é a média harmônica de precisão e recall. A equação que define F1 é a seguinte:
F1 = 2 * Precisão * Recall / (Precisão + Recall).
O valor mais alto possível de uma medida F1 é 1, indicando precisão e recall perfeitos, e o valor mais baixo possível é 0. Quanto maior o valor da medida F1, melhor é o modelo.

E. Matriz de Confusão

A matriz de confusão é uma tabela que permite avaliar o desempenho do classificador utilizado. Cada linha da matriz contém a previsão para uma classe, enquanto cada coluna mostra os valores para a classe real. Neste estudo, a precisão final do modelo é estimada a partir da matriz de confusão de cada iteração do processo de validação cruzada, que são somadas, calculada a média e usada na biblioteca PyCM [27] para calcular as outras métricas.

F. Coeficiente de Correlação de Matthews

O coeficiente de correlação de Matthews (MCC) é essencialmente um coeficiente de correlação entre as classificações binárias observadas e previstas. Varia entre -1 e +1. Um coeficiente de +1 representa uma predição perfeita, 0 (zero) significa que a predição não foi melhor do que a predição aleatória e -1 indica uma discordância total entre predição e observação. O MCC leva em consideração verdadeiros positivos e falsos negativos e, geralmente, é considerada uma medida equilibrada que pode ser usada mesmo se as classes forem de tamanhos muito diferentes [28].

G. Características Operacionais do Receptor

As características Operacionais do Receptor *(Receiver Operating Characteristics - ROC)*, é uma análise que permite comparar a taxa de verdadeiros positivos e a taxa de falsos positivos. A área sob a curva de característica de operação do receptor (AUC) fornece uma medida agregada de desempenho em todos os limites de classificação possíveis. ROC varia em valor de 0 a 1 [26]. Um modelo cujas previsões são 100% erradas tem um ROC de 0,0; aquele cujas previsões são 100% corretas tem um ROC de 1.0. Esta é, também, uma métrica interessante para tarefas com classes não balanceadas [28].

H. Micro e Macro Médias (Micro and Macro Average)

A macro média é simplesmente a média aritmética das medições para cada classe. Por outro lado, a micro média considera o peso de cada classe. Portanto, as classes maiores têm mais influência na média calculada.

IV. RESULTADOS E ANÁLISES

Nesta seção, analisaremos os resultados obtidos. O objetivo principal é responder às perguntas motivadoras e investigativas formuladas nas Seções 1 e 2.

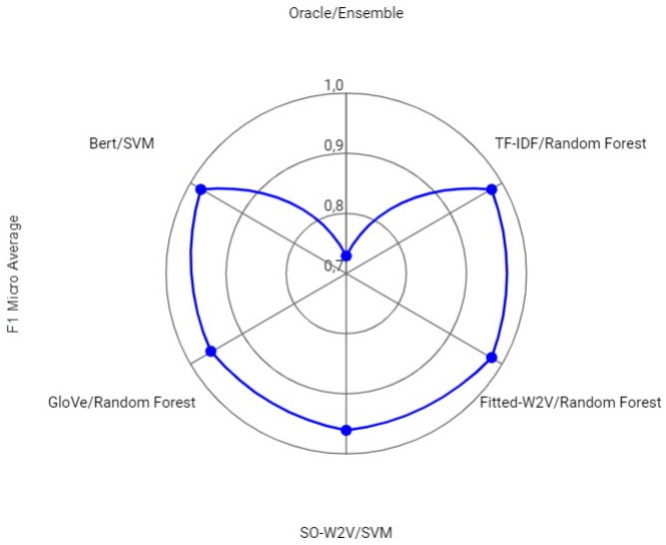

Figura 3: Performance Classificador por Modelo Linguagem

A. *Pergunta Motivadora*

A classificação de artefatos de software terá melhores resultados com a adoção das práticas de PLN?

Conduzimos experimentos com dois modelos de linguagem diferentes para comparar com a abordagem proposta no trabalho de referência [1]. Fizemos o pré-processamento dos dados seguindo as técnicas de PLN, que foram: limpeza dos dados, reamostragem aleatória, transformação dos dados e redução de dimensionalidade. No fluxo dos experimentos, realizamos vetorização (dois processos distintos, um para modelo de espaço vetorial e outro para a representação distribuída). Em seguida, os parâmetros dos classificadores foram ajustados para obter melhor desempenho. Para avaliar os resultados, 5 métricas foram adotadas: precisão, recall, F1, MCC e ROC. Cada classificador foi executado 10 vezes, usando 10 vez a validação cruzada, usamos a média das execuções para calcular todas as demais métricas. Como conclusão, podemos afirmar que o uso das técnicas de PLN com a adoção do modelo de espaço vetorial e a representação distribuída, propiciou desempenho 34% (F1 Micro Média) superior a abordagem manual de extração de atributos.

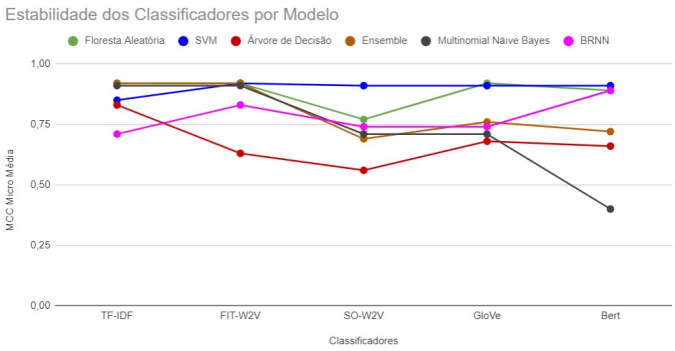

Figura 4: Análise da estabilidade dos classificadores - MCC

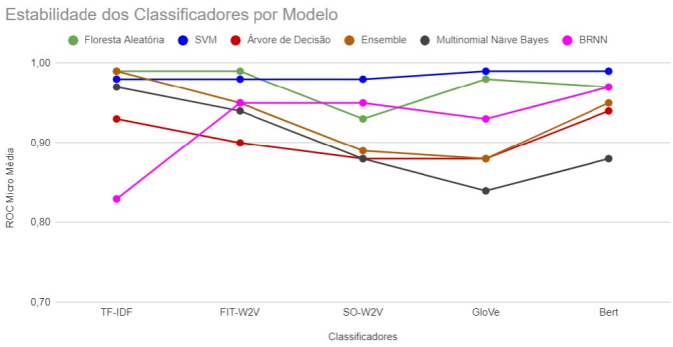

Figura 5: Análise da estabilidade dos classificadores - ROC

Avaliamos o comportamento dos classficadores nos diversos vetores utilizados. Primeiramente, do ponto de vista do MCC Micro Média, o SVM se mostrou o mais estável, alcançando coeficiente mínimo de 0,85 com o TF-IDF, e máximo de 0,92 com o Fitted-Word2Vec. Uma variação inferior a 10% entre os modelos. O segundo classificador mais estável foi a Floresta Aleatória, obtendo mínimo de 0,77 com o SO-Word2Vec e 0,92 de máximo com o TF-IDF, Fitted-Word2VEc e GloVe, variando 19,5%. No geral, as medidas do MCC foram estáveis, indicando que todos os classificadores realizaram bem suas tarefas, exceção ao Multinomial Naive Bayes com o Bert, que obteve coeficiente de 0,40. Do ponto de vista do índice ROC, também observamos o SVM com os melhores indicadores, obtendo 0,98 de mínimo e 0,99 de máximo. Uma variação irrelevante, confirmando a sua grande estabilidade em todos os modelos. A Floresta Aleatória teve o segundo melhor desempenho, obtendo índice mínimo de 0,93 e máximo de 0,99, representando uma variação de 6%. Na prática, também foi um classificador muito estável em todos os modelos avaliados. Veja os gráficos de MCC e ROC nas figuras 4 e 5.

B. *PI1 - Qual classificador obterá melhor desempenho e em qual vetorização?*

Para responder a essa pergunta, comparamos o desempenho dos classificadores selecionados. Cada classificador foi usado com os vetores TF-IDF, Fitted-Word2Vec, SO-Word2Vec, GloVe e Bert. Como nosso objetivo central era identificar a combinação de algoritmo de classificação e vetor mais performático, destacamos na Tabela I os classificadores/vetores que obtiveram as melhores avaliações. Como se pode verificar, os melhores indicadores foram obtidos com a Floresta Aleatória e SVM, ambos alcançando F1 Micro Média de 0,98. No caso da Floresta Aleatória, com TF-IDF e Fitted-Word2Vec. Já o SVM obteve o seu melhor desempenho com o Bert. A Floresta Aleatória (com TF-IDF e Fitted Word2Vec), também obteve as melhores medidas F1 Micro Média para as 7 categorias de artefatos de software. Em todas as categorias, os valores acima de 0,90 para MCC e ROC indicam que o classificador funcionou muito bem no conjunto de dados de validação. Os valores na Tabela II, mostram os maiores valores obtidos de F1 Macro Média para cada categoria. Em geral, a classificação com os modelos de espaço vetorial e representação distribuída indicam empate técnico. O gráfico

Tabela I: Melhor classificador por modelo de linguagem.

Modelo	Classificador	Parâmetros	Categoria	Precisão	Recall	F1	MCC	ROC
Oracle [1]	Ensemble	Default	Documento de Projeto	**1,00**	0,40	0,57	0,62	0,70
			Lista de Colaboradores	0,86	1,00	0,92	0,92	0,99
			Arquivos de Configuração	0,75	0,90	0,82	0,78	0,92
			Documento de Requisito	0,33	0,14	0,20	0,15	0,55
			Licença	1,00	0,90	0,95	0,94	0,95
			Guia do Colaborador	0,90	0,82	0,86	0,83	0,90
			Notas de Versão	0,50	0,80	0,62	0,54	0,82
			Micro Média	0,76	0,75	0,73	0,70	0,85
			Macro Média	0,76	0,71	0,70	0,68	0,83
TF-IDF	Floresta Aleatória	500 árvores	Documento de Projeto	1,00	1,00	1,00	1,00	1,00
			Lista de Colaboradores	1,00	1,00	1,00	1,00	1,00
			Arquivos de Configuração	1,00	0,88	0,93	0,92	0,94
			Documento de Requisito	1,00	1,00	1,00	1,00	1,00
			Licença	1,00	1,00	1,00	1,00	1,00
			Guia do Colaborador	0,88	1,00	0,93	0,92	0,99
			Notas de Versão	1,00	1,00	1,00	1,00	1,00
			Micro Média	0,98	0,98	0,98	0,92	0,99
			Macro Média	0,98	0,98	0,98	0,98	0,99
Fit-Word2Vec	Floresta Aleatória	500 árvores	Documento de Projeto	1,00	1,00	1,00	1,00	1,00
			Lista de Colaboradores	1,00	1,00	1,00	1,00	1,00
			Arquivos de Configuração	1,00	0,88	0,93	0,92	0,94
			Documento de Requisito	1,00	1,00	1,00	1,00	1,00
			Licença	1,00	1,00	1,00	1,00	1,00
			Guia do Colaborador	0,88	1,00	0,93	0,92	0,99
			Notas de Versão	1,00	1,00	1,00	1,00	1,00
			Micro Média	0,98	0,98	0,98	0,92	0,99
			Macro Média	0,98	0,98	0,98	0,98	0,99
SO-Word2Vec	SVM	Default	Documento de Projeto	1,00	1,00	1,00	1,00	1,00
			Lista de Colaboradores	1,00	1,00	1,00	1,00	1,00
			Arquivos de Configuração	1,00	0,86	0,92	0,91	0,93
			Documento de Requisito	0,83	1,00	0,91	0,90	0,99
			Licença	1,00	1,00	1,00	1,00	1,00
			Guia do Colaborador	1,00	0,88	0,93	0,92	0,94
			Notas de Versão	0,86	1,00	0,92	0,91	0,99
			Micro Média	0,96	0,96	0,96	0,91	0,98
			Macro Média	0,96	0,96	0,96	0,95	0,98
GloVe 42B300d	Floresta Aleatória	500 árvores	Documento de Projeto	1,00	1,00	1,00	1,00	1,00
			Lista de Colaboradores	1,00	1,00	1,00	1,00	1,00
			Arquivos de Configuração	1,00	0,88	0,93	0,92	0,94
			Documento de Requisito	0,83	1,00	0,91	0,90	0,99
			Licença	1,00	1,00	1,00	1,00	1,00
			Guia do Colaborador	0,88	0,88	0,88	0,85	0,93
			Notas de Versão	1,00	1,00	1,00	1,00	1,00
			Micro Média	0,96	0,96	0,96	0,92	0,98
			Macro Média	0,96	0,96	0,96	0,95	0,98
Bert Básico	SVM	Default	Documento de Projeto	0,98	0,96	0,97	0,97	0,98
			Lista de Colaboradores	1,00	1,00	1,00	0,99	1,00
			Arquivos de Configuração	1,00	0,82	0,90	0,91	0,91
			Documento de Requisito	0,98	0,99	0,98	0,98	0,99
			Licença	1,00	1,00	1,00	1,00	1,00
			Guia do Colaborador	0,89	0,91	0,90	0,89	0,95
			Notas de Versão	0,96	0,98	0,97	0,97	0,99
			Micro Média	0,98	0,98	0,98	0,91	0,99
			Macro Média	0,97	0,95	0,96	0,96	0,97

Tabela II: Melhor desempenho das categorias por modelo.

Categoria	**Oracle**	**TF-IDF**	**Fitted-W2V**	**SO-W2V**	**GloVe**	**Bert**
Documento de Projeto	0,57	1,00	1,00	1,00	1,00	0,97
Lista de Colaboradores	0,92	1,00	1,00	1,00	1,00	1,00
Arquivos de Instalação	0,82	0,93	0,93	0,92	0,93	0,90
Documento de Requisito	0,20	1,00	1,00	0,91	0,91	0,98
Licença	0,95	1,00	1,00	1,00	1,00	1,00
Guia do Colaborador	0,86	0,93	1,00	0,93	0,88	0,90
Notas de Versão	0,62	1,00	1,00	0,92	1,00	0,97
F1 Macro Média	**0,71**	**0,98**	**0,99**	**0,95**	**0,96**	**0,96**

da figura 7 apresenta o melhor desempenho para cada classificador e vetorização utilizada.

C. PI2 - Na representação distribuída, um vetor de palavra de domínio específico terá melhor desempenho do que um vetor de palavras genérico?

Pelos resultados obtidos nos experimentos realizados, não observamos vantagem nos resultados quando usamos

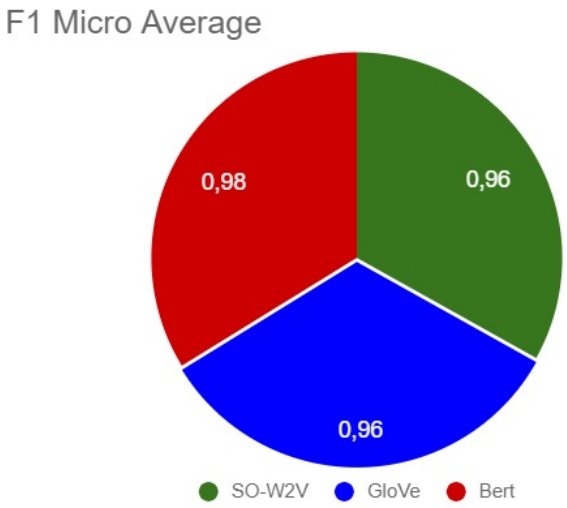

Figura 6: Vetor Específico x Vetor Genérico

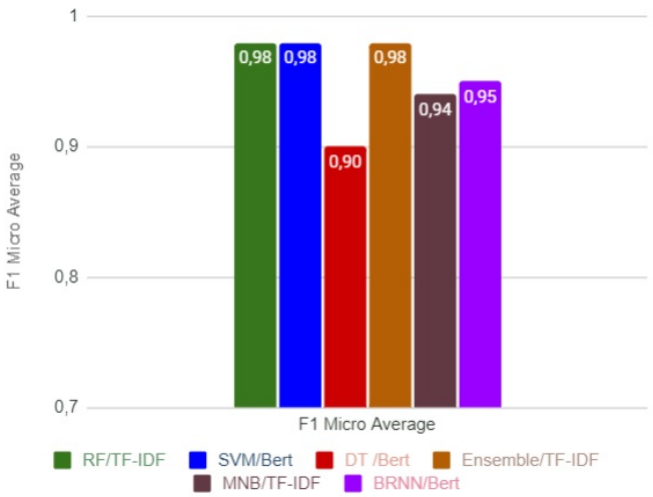

Figura 7: Melhor desempenho por classificador e modelo de linguagem.

um vetor de palavras gerado a partir de um corpus no mesmo domínio de aplicação em que trabalhamos (ou seja, domínio da engenharia de software). Para investigar esse problema, executamos experimentos com o SO-Word2Vec [3]. Os resultados obtidos por este vetor, superaram em 31,5% os resultados obtidos pela abordagem de [1] (F1 Micro Média como parâmetro comparativo). Quando realizamos os experimentos usando os vetores genéricos, obtivemos resultado ligeiramente superior. No caso do Bert (F1 Micro Média de 0,98 com o classificador SVM) e resultado igual com o GloVe (F1 Micro Média de 0,96). Veja o resumo no gráfico de pizza da figura **??**. Esta conclusão é muito relevante, visto que existem vários vetores de palavras pré-treinados disponíveis e, portanto, a um baixo custo que podem ser utilizados nas tarefas de classificação de textos, e neste caso particular, classificação de artefatos de software, sendo possível obter alto desempenho sem a necessidade de demandar investimento em um vetor de palavras específico e que, teoricamente, demandaria mais recursos financeiros para sua obtenção.

V. CONCLUSÕES E OPORTUNIDADES

Este artigo apresenta um estudo comparativo da classificação de artefatos de software, com o uso do aprendizado de máquina, entre três abordagens distintas: a primeira abordagem faz a identificação manual dos atributos textuais para que possam ser submetidos aos algoritmos de classificação. Essa abordagem foi realizadas por especialista no domínio de engenharia de software (oráculo) [1]. A segunda abordagem, faz uso das técnicas de PLN para o pré-processamento dos dados, seleção e extração dos atributos, adotando o modelo de espaço vetorial com TF-IDF para a vetorização. A terceira abordagem, também faz uso da PLN, porém adota a representação distribuída, em que vetores densos de palavras *(word embeddings)* são usados para transferir conhecimentos semânticos aos atributos existentes e depois se realiza a vetorização para que os classificadores executem a sua tarefa. Verificamos que as abordagens que se utilizaram das técnicas de PLN, com o uso de extração automática de atributos e os modelos de espaço vetorial e representação distribuída obtiveram melhores resultados na classificação automática de artefatos de software, superando em 34% (F1 Micro Média) o modelo com extração manual de atributos, proposto no estudo de referência. Três combinações de classificadores e vetores, alcançaram 0,98 de F1 Micro Média. Foram eles: Floresta Aleatória com os vetores TF-IDF e Fitted-Word2Vec, SVM com o Bert e o classificador *Ensemble* com TF-IDF. Não menos relevantes, foram os resultados obtidos com o SVM fazendo uso do vetor SO-Word2Vec e Floresta Aleatória com o GloVe que, em ambos os casos, alcançaram F1 Micro Média de 0,96. Concluímos, em ambas as abordagens propostas neste artigo, que o desempenho foi superior ao obtido no trabalho seminal. Além disso, podemos ver que os *embeddings* de propósito geral foram suficientes para obter o melhor desempenho na tarefa de classificação proposta.

Como contribuição para pesquisas futuras, este artigo criou e disponibilizou um repositório no github com os códigos-fonte e tabelas com os resultados obtidos (https://github.com/mauriciobritojr/Software-Artifacts-Classification). Como oportunidades futuras de estudos, vemos a possibilidade de aumentar a amostra de dados e a utilização de outras técnicas de classificação visando maior precisão, principalmente, utilizando novas arquiteturas de redes neurais e vetores de palavras com um maior número de parâmetros gerados, recentemente.

REFERÊNCIAS

[1] Y. Ma, S. Fakhoury, M. Christensen, V. Arnaoudova, W. Zogaan, and M. Mirakhorli, “Automatic classification of software artifacts in open-

source applications," in *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*. IEEE, 2018, pp. 414–425.
[2] K. S. Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of documentation*, 1972.
[3] V. Efstathiou, C. Chatzilenas, and D. Spinellis, "Word embeddings for the software engineering domain," in *Proceedings of the 15th international conference on mining software repositories*, 2018, pp. 38–41.
[4] F. N. A. Al Omran and C. Treude, "Choosing an nlp library for analyzing software documentation: a systematic literature review and a series of experiments," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 187–197.
[5] R. Řehřek, "Scalability of semantic analysis in natural language processing," Ph.D. dissertation, Masarykova univerzita, Fakulta informatiky, 2011.
[6] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.
[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.
[8] B. Bengfort, R. Bilbro, and T. Ojeda, *Applied text analysis with python: Enabling language-aware data products with machine learning*. "O'Reilly Media, Inc.", 2018.
[9] N. Japkowicz *et al.*, "Learning from imbalanced data sets: a comparison of various strategies," in *AAAI workshop on learning from imbalanced data sets*, vol. 68. AAAI Press Menlo Park, CA, 2000, pp. 10–15.
[10] G. E. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM SIGKDD explorations newsletter*, vol. 6, no. 1, pp. 20–29, 2004.
[11] S. Kotsiantis, D. Kanellopoulos, P. Pintelas *et al.*, "Handling imbalanced datasets: A review," *GESTS International Transactions on Computer Science and Engineering*, vol. 30, no. 1, pp. 25–36, 2006.
[12] A. Vabalas, E. Gowen, E. Poliakoff, and A. J. Casson, "Machine learning algorithm validation with a limited sample size," *PloS one*, vol. 14, no. 11, p. e0224365, 2019.
[13] A. I. Kadhim, "An evaluation of preprocessing techniques for text classification," *International Journal of Computer Science and Information Security*, vol. 16, no. 6, 2018.
[14] Y. Yang, "An evaluation of statistical approaches to text categorization," *Information retrieval*, vol. 1, no. 1, pp. 69–90, 1999.
[15] G. Kou, P. Yang, Y. Peng, F. Xiao, Y. Chen, and F. E. Alsaadi, "Evaluation of feature selection methods for text classification with small datasets using multiple criteria decision-making methods," *Applied Soft Computing*, vol. 86, p. 105836, 2020.
[16] V. B. Kobayashi, S. T. Mol, H. A. Berkers, G. Kismihok, and D. N. Den Hartog, "Text classification for organizational researchers: A tutorial," *Organizational research methods*, vol. 21, no. 3, pp. 766–799, 2018.
[17] V. V. Raghavan and S. M. Wong, "A critical analysis of vector space model for information retrieval," *Journal of the American Society for information Science*, vol. 37, no. 5, pp. 279–287, 1986.
[18] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *International conference on machine learning*. PMLR, 2014, pp. 1188–1196.
[19] R. Pappagari, P. Zelasko, J. Villalba, Y. Carmiel, and N. Dehak, "Hierarchical transformers for long document classification," in *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 2019, pp. 838–844.
[20] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.
[21] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.
[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.
[23] T. G. Dietterich *et al.*, "Ensemble learning," *The handbook of brain theory and neural networks*, vol. 2, pp. 110–125, 2002.
[24] I. Kukenys and B. McCane, "Classifier cascades for support vector machines," in *2008 23rd International Conference Image and Vision Computing New Zealand*. IEEE, 2008, pp. 1–6.
[25] P. Liu, X. Qiu, and X. Huang, "Recurrent neural network for text classification with multi-task learning," *arXiv preprint arXiv:1605.05101*, 2016.
[26] D. M. Powers, "Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation," *arXiv preprint arXiv:2010.16061*, 2020.
[27] S. Haghighi, M. Jasemi, S. Hessabi, and A. Zolanvari, "Pycm: Multiclass confusion matrix library in python," *Journal of Open Source Software*, vol. 3, no. 25, p. 729, 2018.
[28] S. Boughorbel, F. Jarray, and M. El-Anbari, "Optimal classifier for imbalanced data using matthews correlation coefficient metric," *PloS one*, vol. 12, no. 6, p. e0177678, 2017.