Classificação de Issues do Github com Relação a Segurança

1st Bruno Gonçalves de Oliveira
Universidade Federal do Paraná (UFPR)
Curitiba– PR – Brasil
bruno.mphx2@gmail.com

2nd Diogo Cezar Teixeira Batista
Universidade Federal do Paraná (UFPR)
Curitiba– PR – Brasil
diogocezar@utfpr.br

***Abstract*—As *issues* do GitHub representam grande parte da evolução dos projetos *OpenSource*. Este trabalho propõe uma solução para classificar *issues* que podem ou não estar relacionadas a segurança da informação. As mensagens serão analisadas utilizando as palavras frequentes em textos referentes à segurança. Utiliza-se a técnica de *Bag-of-Words* em conjunto com *TF-IDF* para a criação dos vetores de representação. Na sequência, múltiplos classificadores foram utilizados, entre eles: svm, knn, naive_bayes, lda, logistic_regression, perceptron, tree e mlp. Os resultados obtidos apresentam uma acurácia superior a 80%.**

***Index Terms*—Issues, GitHub, Segurança da Informação, Classificadores, Aprendizagem de Máquina**

I. Introdução

O controle, gerenciamento e manutenção de arquivos, especialmente no âmbito do desenvolvimento de *software*, sempre foi um desafio. Problemas recorrentes como: *backups* não realizados, sobrescrita de arquivos ou dificil manutenabilidade de projetos desenvolvidos em equipes, são algumas das motivações para a utilização de algum sistema de versionamento de arquivos. [1]

Dentre as várias ferramentas existenstes no mercado, tais como: *CVS*, *Subversion*, *TFS*, *Mercurial*, o *Git* se destaca por ter sido amplamente utilizado pela comunidade de desenvolvimento, com o advento da popularização dos projetos *OpenSource*. Estes projetos se consolidaram em ferramentas como o *GitHub* que é uma plataforma para versionamento, gerenciamento e colaboração de projetos, que utiliza o *Git* como base. [1]

São várias as possibilidades que essas ferramentas proporcionam para verionamento dos projetos, no *GitHub*, por exemplo, existe uma sessão de *issues* (problemas) na qual, os colaboradores de um projeto *Open Source* podem cadastrar possíveis *bugs*, melhorias, ou novas *features* para os projetos.

Na descrição dessas *issues* feita através do preenchimento de um campo de texto, é possível identificar qual é o contexto em que se aplica determinada correção, melhoria ou *feature*.

Eventualmente, dentre as correções realizadas nestes projetos, são identificados ajustes relacionados a segurança. Estas *issues*, quando consideradas críticas, podem ser analisadas por outros especialistas para uma arguição mais detalhada e um possível aprimoramento.

Mas como identificar quais são os ajustes de um projeto que estão relacionados com segurança? Como separar estes ajustes para que especialistas possam analisar os cógidos? Essas perguntas guiam a motivação para o desenvolvimento deste trabalho.

Propõe-se a criação de uma ferramenta que utilize técnicas de aprendizagem de máquina para a criação de um oráculo classificador que consiga analisar as palavras contidas em nas mensagens das *issues* de um dado projeto, e classificar se esta *issue* está ou não relacionada à alguma implementação relacionada com segurança da informação.

Por fim, discute-se os resultados obtidos dos experimentos e possíveis trabalhos futuros.

II. Obtenção dos Dados

Projetos *OpenSource* são, por essência, públicos na Internet, e por este motivo, suas *issues* também estão dispostas de forma pública. A análise dessas mensagens faz parte da evolução dos projetos, nos quais os colaboradores podem entender os problemas, sugerir, implementar e finalizar os problemas encontrados.

A Figura 1 mostra um exemplo de algumas issues do projeto Vue.js.

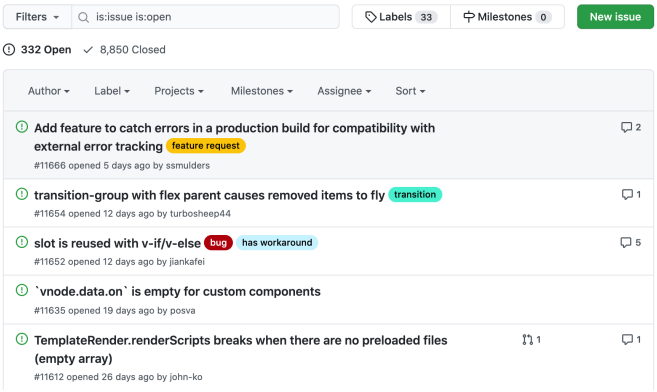

Fig. 1. Exemplos de Issues do Projeto Vue.js

Para a extração dos dados das *issues*, tanto para a base de treinamento quanto de para a base de testes, utilizou-se a a ferramenta *github-csv-tools*[1] que possibilita a exportação dos dados de um repositório no *GitHub*, e os salva em um arquivo no formato CSV.

[1]https://github.com/gavinr/github-csv-tools

Estes dados foram organizados em arquivos no formato CSV que possuem 2 colunas. A primeira coluna indica se a *issue* é um tópico relacionado a segurança ou não, e a segunda coluna representa de fato o texto coletado.

Um exemplo abreviado de cada classe pode ser vista no Código 1.

```
1  security,PushObserver can be used to push
       serverinitiated HTTP/2 requests into an
       OkResponseCache...
2  not,Handle LOCKED in conversions.Motivation...
```

Código 1. CSV Exemplo com Base de Dados

Para a geração da base de testes, foram extraídas as *issues* do projeto *Wildfly*[2]. Já para a base de treinamento utilizou-se *issues* dos projetos: *https://github.com/square/okhttp/*, *https://github.com/eclipse/jgit* e *https://github.com/couchbase*.

Os dados de treinamento possuem **199** entradas, enquanto que para a base de teste foram utilizadas **211** entradas.

III. Pré-processamento

Após a obtenção dos dados, é importante a realização de algumas etapas de pré-processamento do texto, estes tratamentos procuram maximizar a representatividade das *issues* de acordo com o seu significado.

Para isso, é necessário remover palavras que não representam o contexto das frases. Com esse objetivo, aplicou-se técnicas para cada uma das frases das *issues* da base de treinamento e testes.

As regras aplicadas foram:

1) Transformar todo texto em minúsculo;
2) Ignorar pontuações;
3) Corrigir palavras com ortografia incorreta;
4) Remover as chamadas *stop words* que não acrescentam informação aos textos, por exemplo: *of, a, in, on*.

IV. Extração de Características

Para a extração de características das bases de dados, utiliza-se uma técnica conhecida como *Bag-of-Words* **Adicionar Referência**. Nesta técnica, as sentenças são representadas através da identificação de suas palavras e a quantidade em que aparecem no texto. Por exemplo, considerando o seguinte texto, escrito por *Charles Dickens*:

> It was the best of times,
> It was the wost of times,
> It was the age of wisdom,
> It was the age of follishness.

Considerando palavras únicas, neste caso, teríamos um vocabulário com 10 palavras: [it, was, the, best, of, times, worst, age, wisdom, foolishness]

Na sequência deve-se criar os vetores que representam a quantidade de aparições de uma palavra no texto. Por exemplo, no documento "It was the age of wisdom" pode-se representar através de um vetor dado por: $[1, 1, 1, 0, 1, 0, 0, 1, 1, 0]$. Se novas frases, com palavras fora do vocabulário definido forem adicionadas, então essas serão descartadas.

Com os vetores de palavras formados com a identificação e a quantidade, é possível aplicar a técnica de *TF-IDF (term frequency-inverce document frequency)*. Nesta abordagem, valoriza-se o quão importante uma palavra é ao documento. O valor de *TF-IDF* aumenta proporcionalmente conforme o número de vezes a palavra aparece no texto e é compensado pelo número de textos na base de dados, ajustando o número da frequência das palavras. **Adicionar Referência**

Basicamente para cada uma das *issues* já sanitizadas aplica-se uma contagem das palavras mais relevantes para o problema em questão. Essas palavras foram obtidas a partir de uma contagem das palavras que mais apareciam nas *issues* que eram relacionadas à segurança.

Obteve-se as seguintes palavras: ['security', 'secure', 'vulnerable', 'leak', 'exception', 'crash', 'malicious', 'sensitive', 'user', 'authentication', 'protect', 'vulnerability', 'authenticator', 'auth', 'npe']

Após a aplicação destas técnicas, obtém-se 4 vetores que serão utilizados nas próximas fases do experimento. São eles: (x_train, y_train, x_test, y_test)

V. Orquestrador dos Exerimentos

Essencialmente em problemas que podem envolver diferentes classificadores, é bastante comum a realização dos experimentos em diferentes abordagens, utilizando classificadores diferentes, e para cada um dos classificadores, parâmetros diferentes.

Com o intúito de automatizar o processo de variação entre os experimentos, desenvolveu-se um sistmea orquestrador de classificadores.

Este sistema utiliza um arquivo no forma *JSON* para definir 2 principais blocos: configurações e experimentos.

No bloco de configurações, é possível definir quais serão os arquivos de entrada e saída para a realização dos experimentos. Já no bloco de experimentos define-se um *array* com todos os experimentos a serem realizados.

Um exemplo do arquivo de orquestração pode ser visto no Código 2.

```
1  {
2      "configs": {
3          "train": "data/train.csv",
4          "test": "data/test.csv",
5      },
6      "experiments": [
7          {
8          "classifier": "logistic_regression",
9          "parameters": {"solver": "lbfgs"}
10         },
11         {
12         "classifier": "knn",
13         "parameters": {"n_neighbors": 7}
14         },
15         ...
16     ]
17 }
```

Código 2. JSON do Orquestrador

Para cada uma das ocorrências no vetor de experimentos, o programa desenvolvido utiliza implementações do *framework*

[2]https://github.com/wildfly/wildfly

[3]*scikit-learn* para o treinamento do classificador. Utlizando como fonte de dados, as entradas já tratados anteriormente (x_train, y_train, x_test, y_test). Os resultados são armazenados automaticamente em um arquivo no formato CSV que tabula os resultados obtidos em cada um dos experimentos. Os campos salvos são: *Classifier, F1Score, Accuracy e Time (s)* Além disso, para cada execução são armazenadas as matrizes de confusão como imagens e também como arquivos CSV.

VI. Resultados Obtidos

O orquestrador dos experimentos foi configurado para executar os experimentos com os classificadores: *svm.LinearSVC KNeighborsClassifier*, *GaussianNB*, *LinearDiscriminantAnalysis*, *LogisticRegression*, *Perceptron*, *DecisionTreeClassifier* e *MLPClassifier*.

Os parâmetros utilizados foram os default em cada um dos classificadores. Com a exeção do *KNeighborsClassifier* em que se testou com *n_neighbors: 7* e *LogisticRegression* que se utilizou *solver: lbfgs*.

Os resultados obtidos estão dispostos na Tabela I.

Classifier	Accuracy	F1Score	Time (s)
LinearDiscriminantAnalysis	0.867	0.865	0.183
LogisticRegression	0.867	0.865	0.191
DecisionTreeClassifier	0.867	0.865	0.193
MLPClassifier	0.867	0.865	0.302
svm.LinearSVC	0.867	0.865	1.903
Perceptron	0.862	0.861	0.17
KNeighborsClassifier	0.533	0.696	0.229
GaussianNB	0.471	0.307	0.192

TABLE I
Resultados dos Experimentos

As matrizes de confusão dos classificadores: *DecisionTreeClassifier*, *GaussianNB KNeighborsClassifier* e *LinearDiscriminantAnalysis* estão dispostas na Figura 2. Já para os classificadores: *svm.LinearSVC*, *LogisticRegression*, *MLPClassifier* e *Perceptron* as representações estão dispostas na Figura 3.

VII. Conclusão

Como é possível observar na Tabela I os classificadores que utilizaram os algorítmos: *LinearDiscriminantAnalysis*, *LogisticRegression*, *DecisionTreeClassifier*, *MLPClassifier*, *svm.LinearSVC* e *Perceptron* tiveram resultados bastante semelhantes, com uma taxa de acurácia em torno de 0.86 e um f1score também por volta de 0.86. A duração do processo de classificação (anotada pela coluna Time (s)) demostram que para o volume de dados utilizado todos tem um tempo de treinamento bem baixo, retornando o resultado em menos de 2 segundos no pior caso.

Para o classificador *KNeighborsClassifier* obteve-se um resultado bastante insatisfatório, com acurácia de 0.53 e f1score de 0.69. Isso se deve **completar...**

Já para o classificador *GaussianNB* também obteve-se um resultado bastante insatisfatório, com acurácia de 0.47 e f1score de 0.3., e isso se deve **completar...**

[3]https://scikit-learn.org/

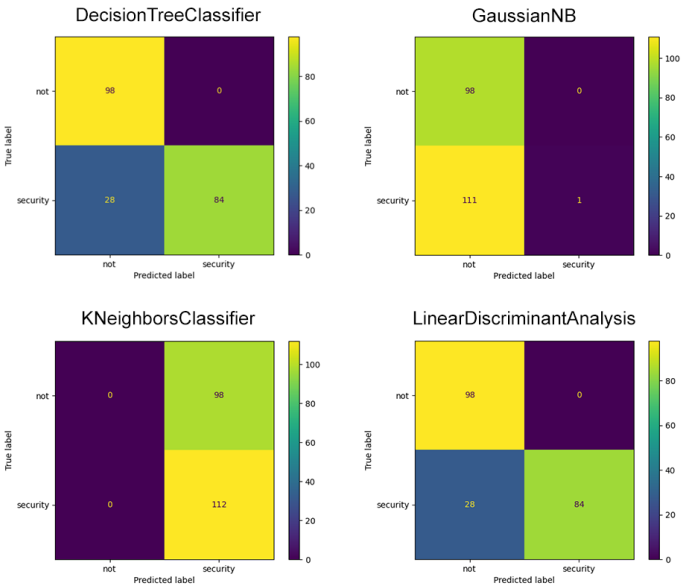

Fig. 2. Matrizes de Confusão 1

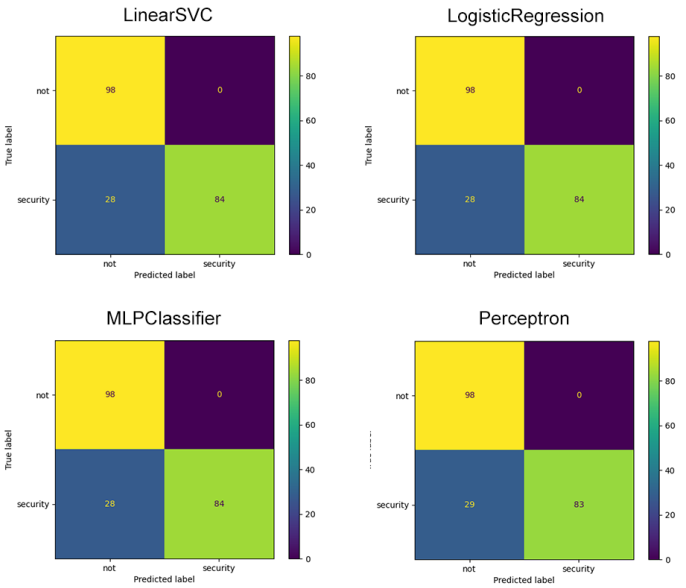

Fig. 3. Matrizes de Confusão 2

Com relação às matrizes de confusão da Figura 2, pode-se notar que para os algorítmos *DecisionTreeClassifier* e *LinearDiscriminantAnalysis* obteve-se a mesma quantidade de erros, e isso se deve **completar...**

Ainda nas matrizes da Figura 2, nota-se que para os piores classificadores (neste problema) *KNeighborsClassifier* e *GaussianNB* houveram vários erros que praticamente anulam a eficiência do classificador.

Com relação às matrizes da Figura 3 é possível notar que praticament o mesmo número de erros foi encontrado, muito

provavelmente nos mesmo exemplos.

VIII. Trabalhos Futuros

A grande dificuldade encontrada no projeto foi a obtenção de uma base de dados rotulada. Toda a base utilizada foi tratada de forma manual.

Para aumentar a base de treinamento de uma forma automatizada, propõe-se no futuro o desenvolvimento de um programa capaz de extrair automaticamente informações de *issues* de repositórios do *GitHub* marcados com um *label* que contenha palavras relacionadas à segurança. O *GitHub*, oferece uma API que possibilita a extração de informações dos projetos no formato JSON, o que facilitaria este processo. É possível notar que em alguns projetos, existem *issues* pré-classificadas com *labels*, como é o exemplo do repositório do projeto Angular, disponível em: https://github.com/angular/angular/labels?q=sec.

Outra estratégia poderia ser a aplicação de técnicas de *Data Augmentation* para possibilitar a criação de características de forma sintéticas, refinando ainda mais os classificadores testados.

IX. Código Fonte

O código fonte dos experimentos pode ser acessado em: https://github.com/bmphx2/aprendizagem-de-maquina

References

[1] S. Chacon and B. Straub, *Pro Git*. Apress, 2020.