

Classificação de *issues* do Github relacionadas 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 a segurança. Utiliza-se a técnica de *Bag-of-Words* em conjunto com *TF-IDF* para a criação do *dataset* de representação. Na sequência, experimentos com classificadores foram realizados para explorar qual a melhor técnica a ser adotada para o problema em questão. Os melhores resultados apresentam uma acurácia superior a 80%, o que demonstra que as soluções exploradas são plausíveis e podem ser melhoradas como é explorado na seção de trabalho futuros.

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 difícil manutenibilidade de projetos desenvolvidos em equipes, são algumas das motivações para a utilização de algum sistema de versionamento de arquivos. [1]

Dentre as várias ferramentas existentes 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 *Open Source*. Tais 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 versionamento 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 códigos? 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 o desenvolvimento de um oráculo classificador que consiga analisar as palavras contidas nas mensagens das *issues* de um dado projeto, e classificar se esta *issue* está ou não relacionada no contexto de segurança da informação.

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

II. OBTENÇÃO DOS DADOS

Projetos *Open Source* 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*.

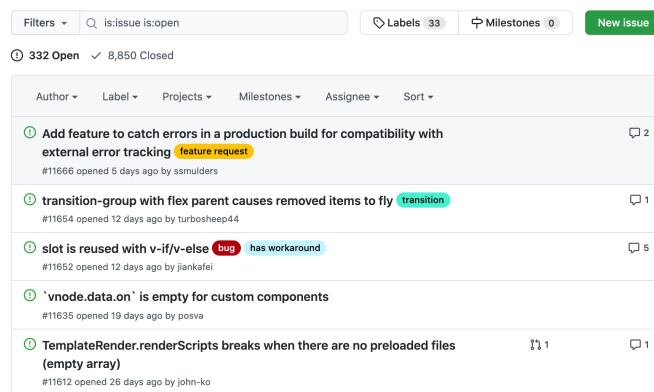


Fig. 1. Exemplos de Issues do Projeto *Vue.js*

Para a extração dos dados das *issues*, tanto para a base de treinamento quanto de para a base de testes, utilizou-se a ferramenta *github-csv-tools*¹ que possibilita a exportação dos

¹<https://github.com/gavinr/github-csv-tools>

dados de um repositório do *GitHub*, salvando as informações em um arquivo no formato CSV.

Os dados foram refinados para compor um arquivo CSV com 2 colunas. A primeira 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, aplica-se técnicas de sanitização para a 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* [2]. 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 worst of times,
It was the age of wisdom,
It was the age of foolishness.

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-inverse 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. [3]

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 EXPERIMENTOS

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 intuito de automatizar o processo de variação entre os experimentos, desenvolveu-se um sistema orquestrador de classificadores.

Este sistema utiliza um arquivo no formato *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. Utilizando 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. CLASSIFICAÇÃO

Para o trabalho foram utilizados 8 classificadores:

- *Linear Discriminant Analysis* (LDA): técnica de transformação linear que maximiza a característica que se pode separar entre classes e minimiza o espalhamento dentro da classe; [4]
- Máquina de Vetores de Suporte *Support Vector Machine* (SVM): o classificador tem como objetivo encontrar um hiperplano que maximiza a separabilidade das classes; [4]
- Vizinhos mais próximos (*KNN*): algoritmo utilizado para classificação que define a classe de uma nova amostra com base na representatividade de um conjunto de k vizinhos mais próximos; [5]
- *Multi-layer Perceptron* (MLP): é uma rede neural com múltiplas camadas compostas de neurônios que realizam o mapeamento das entradas ponderadas em um espaço não linear; [4]
- O *Perceptron* é um tipo de rede neural artificial que representa um classificador binário que mapeia sua entrada x (um vetor de valor real) para um valor de saída $f(x)$ utilizando pesos para aprender ao longo do tempo; [6]
- Uma árvore de decisão (*Decision Tree*) consiste em uma coleção de nós internos e nós folhas, organizados em um modelo hierárquico como uma estrutura de dados, que orienta a classificação (baseado em comparações) de uma amostra a ser classificada. [5]
- O *Multilayer Perceptron* (MLP) é semelhante ao *Perceptron*, porém com a utilização de mais de uma camada para a aprendizagem ao longo do tempo;
- A técnica de mistura de gaussianas é uma abordagem probabilística para realizar um agrupamento do tipo *soft clustering*. Este tipo de abordagem atribui um exemplo a uma classe com uma dada probabilidade; [7]

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 execução do *KNeighborsClassifier* em que se testou com $n_neighbors: 7$ e *LogisticRegression* que se utilizou *solver: lbfgs*.

VII. RESULTADOS OBTIDOS

Esta seção mostra os resultados obtidos após a execução do orquestrador para os diferentes classificadores.

³<https://scikit-learn.org/>

A. Avaliação de Desempenho

Para a medição dos resultados utilizou-se a Acurácia e F1Score.

A Acurácia mede a razão entre o número de previsões corretas e o número total de amostras de entrada e é dada por:

$$\text{Acurácia} = \frac{VP + VN}{VP + VN + FP + FN} \quad (1)$$

onde, VP é os positivos verdadeiros e FN os negativos falsos.

Já a métrica de F1Score, é dada por:

$$\text{F1Score} = 2 \cdot \frac{P \cdot R}{P + R} \quad (2)$$

onde P (precisão) é o número de verdadeiros positivos dividido pelo número de positivos estimados pelo modelo. e R (*recall*) é o número de verdadeiros positivos dividido pela somada $VP + FN$.

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.

VIII. CONCLUSÃO

Como é possível observar na Tabela I os classificadores que utilizaram os algoritmos: *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)) demonstram 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, pois os exemplos provavelmente foram projetados de forma esparça.

Já para o classificador *GaussianNB* também obteve-se um resultado bastante insatisfatório, com acurácia de 0.47 e

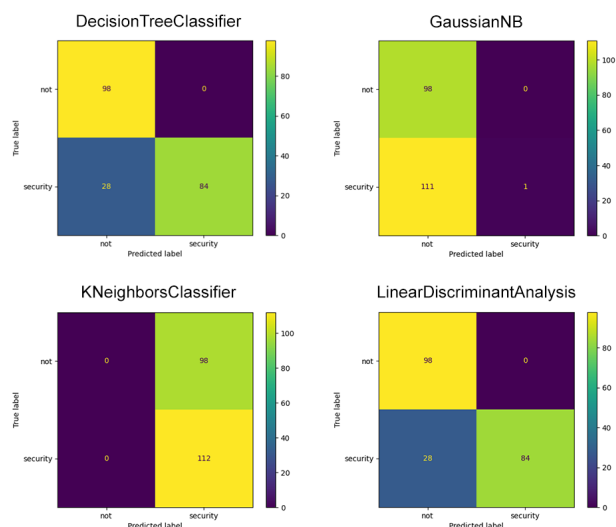


Fig. 2. Matriz de Confusão 1

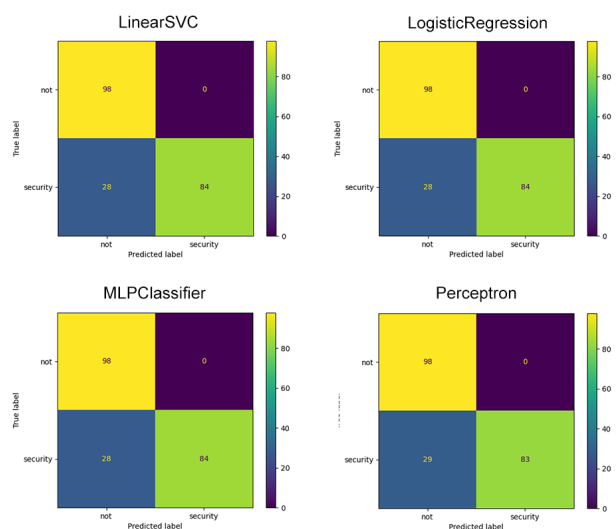


Fig. 3. Matriz de Confusão 2

f1score de 0.3, e isso se deve também pela projeção esparçada dos exemplos.

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 praticamente o mesmo número de erros foi encontrado, muito provavelmente nos mesmos exemplos.

IX. 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. O que dificultou bastante a geração da base de testes e validação.

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.

X. 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.
- [2] D. Mohey El-Din, "Enhancement bag-of-words model for solving the challenges of sentiment analysis," *International Journal of Advanced Computer Science and Applications*, vol. 7, 01 2016.
- [3] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge, UK: Cambridge University Press, 2008. [Online]. Available: <http://nlp.stanford.edu/IR-book/information-retrieval-book.html>
- [4] A. Webb, *Statistical Pattern Recognition: Third Edition*. Ltd. John Wiley and Sons: New York, 2002.
- [5] R. Duda, P. Hart, and D. G. Stork, *Pattern Classification*. Ltd. John Wiley and Sons: New York, 2001.
- [6] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," *Psychological review*, vol. 65, pp. 386–408, 1958.
- [7] A. Pinkus, "Approximation theory of the mlp model in neural networks," *ACTA NUMERICA*, vol. 8, pp. 143–195, 1999.