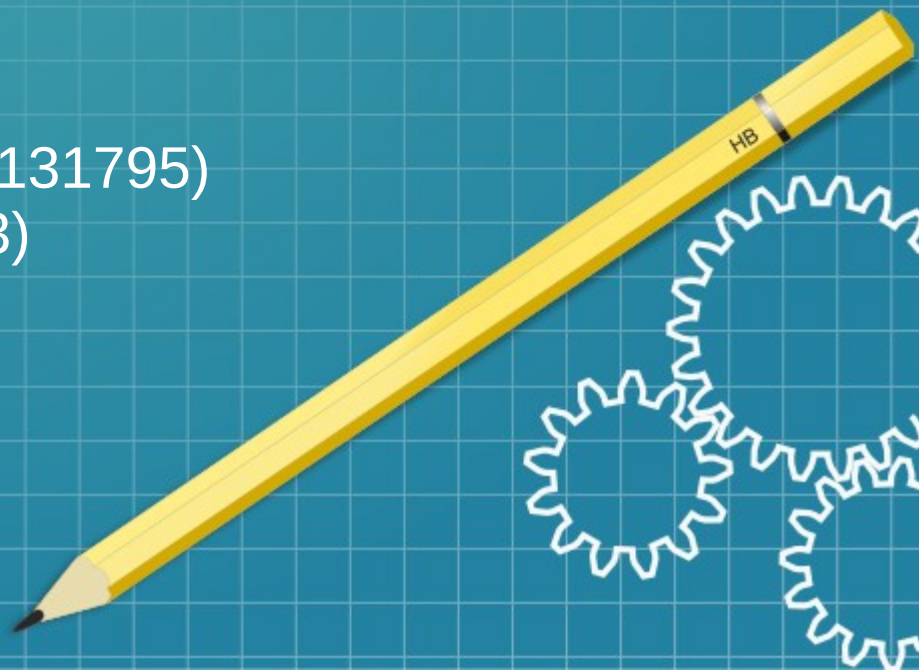
Ciência de Dados para Segurança DInf/UFPR

UFPR – Mestrandos:
Michael A Hempkemeyer (PPGINF-202000131795)
Roger R R Duarte (PPGINF-202000131793)



Objetivo do trabalho

- Dataset relacionado a CVEs foi utilizado (Common Vulnerabilities and Exposures);
- O objetivo traçado foi o mapeamento de quais CVE geraram impacto e quais não geraram sobre determinados tipos de ambientes, isso com base nas variáveis do Dataset;
- São apresentados o Dataset e seus respectivos rótulos, informações a respeito do pré-processamento do Dataset, gráficos distribuição de classes, informações de treinamentos, testes e resultados obtidos com o processamento do Dataset utilizando os algoritmos RandomForest, Kneighbors e SVM.

Dataset

- O dataset possui um arquivo único com diversos JSONs (um por linha) com informações específicas de CVEs;
- Os campos cvss, cwe, access, impact, summary e vulnerable_configuration_cpe_2_2 foram utilizados em nosso trabalho, sendo eliminados os demais através do pré-processamento;

- Acesso ao dataset completo em https://www.kaggle.com/vsathiamoo/cve-common-vulnerabilities-and-exposures/version/1. ;

Dataset - Completo

- Modified | tipo: date
- Published | tipo: date
- Access | tipo: dict { "authentication":
  "MULTIPLE_INSTANCES", "NONE" ou "SINGLE_INSTANCE",
  "complexity":
  "HIGH", "LOW" ou "MEDIUM",
  "vector":
  "ADJACENT_NETWORK", "LOCAL" ou "NETWORK"
}
- Capec | tipo: list() | obs.: Common Attack Pattern Enumeration and Classification (CAPEC™)
- Cvss | tipo: float
- Cvss-time | tipo: date
- Cwe | tipo: string
- id (Cve-id) | tipo: string
- Impact | tipo: dict { "availability":
  "PARTIAL", "COMPLETE" ou "NONE",
  "confidentiality":
  "PARTIAL", "COMPLETE" ou "NONE",
  "integrity":
  "PARTIAL", "COMPLETE" OU "NONE"
}
- last-modified | tipo: date
- Nessus | tipo: list() | obs.: Informação fornecida pelo site www.tenable.com, indica CVEs relacionados
- References | tipo: list()
- Summary | tipo: string
- Vulnerable_configuration | tipo: list() | obs.: configuração do produto vulnerável
- Vulnerable_configuration_cpe_2_2 | tipo: list() | obs.: configuração do produto vulnerável

Pré-processamento

- O pré-processamento foi realizado através do Script Python PreProcessamento.py, de forma a se obter informações de interesse do Dataset;
- As colunas summary e cvss foram utilizadas como base para determinar quais linhas do dataset seriam mantidas, visto que a coluna summary em determinados momentos possuía a mensagem "** REJECT ** DO NOT USE THIS CANDIDATE NUMBER" e a coluna cvss (score) possuía itens em branco.

Pré-processamento

- Com a coluna summary foi possível mapear quais CVEs geraram impacto e quais não geraram (objetivo do trabalho).

```
(...)
elif d == "impact":
        if ((d in tmp.keys()) and
            (tmp[d]["availability"] == "PARTIAL" or tmp[d]["availability"] == "COMPLETE") and
            (tmp[d]["confidentiality"] == "PARTIAL" or tmp[d]["confidentiality"] ==
                                                    "COMPLETE") and
             (tmp[d]["integrity"] == "PARTIAL" or tmp[d]["integrity"] == "COMPLETE")):
                tmp_dict["impact"] = 1
        else:
                tmp_dict["impact"] = 0
(...)
```

Pré-processamento

- Para a coluna access, o seguinte tratamento foi realizado:

```
(...)
self.control_access = {
            "vector": {
                  "ADJACENT_NETWORK": 1,
                  "LOCAL": 2,
                  "NETWORK": 3
             },
            "complexity": {
                  "HIGH": 5,
                  "LOW": 6,
                  "MEDIUM": 7
             },
             "authentication": {
                   "MULTIPLE_INSTANCES": 9,
                   "NONE": 10,
                   "SINGLE_INSTANCE": 11
             },
             "NotAvailable": 12
}
(...)
```

Pré-processamento

```
(...)
elif d == "access":
        if d in tmp.keys():
        # Faz a categorização do access conforme variável self.access_control
                tmp_dict["access"] = self.control_access["vector"][tmp[d]["vector"]]
                tmp_dict["access"] += self.control_access["authentication"]
                                                [tmp[d["authentication"]]
                 tmp_dict["access"] += self.control_access["complexity"][tmp[d]
                                                ["complexity"]]
        else:
                 tmp_dict["access"] = self.control_access["NotAvailable"]
(...)
```

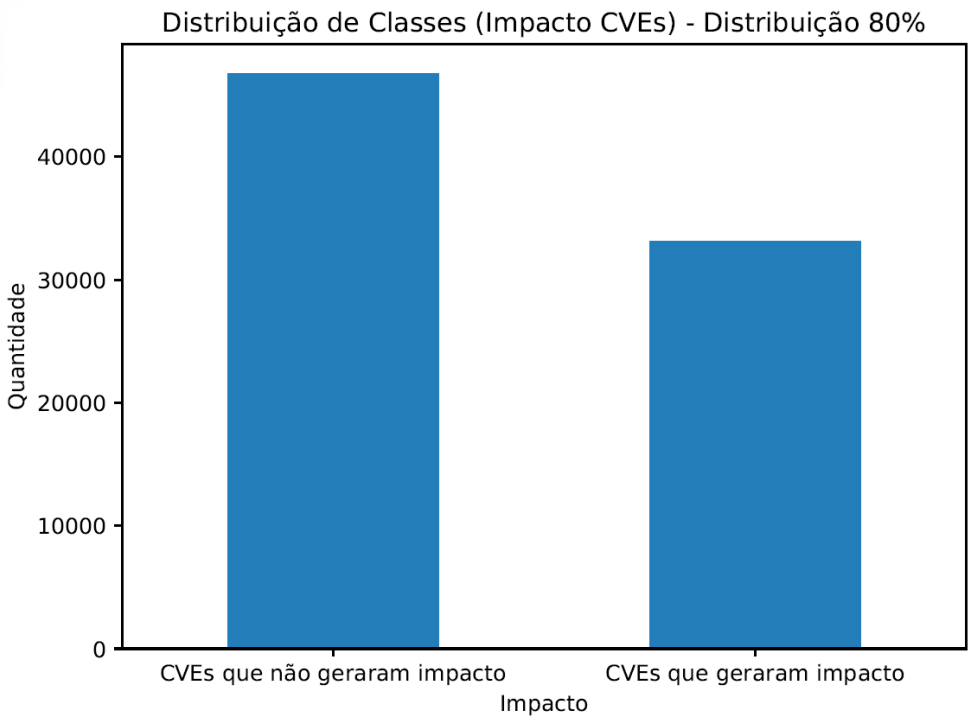
Pré-processamento

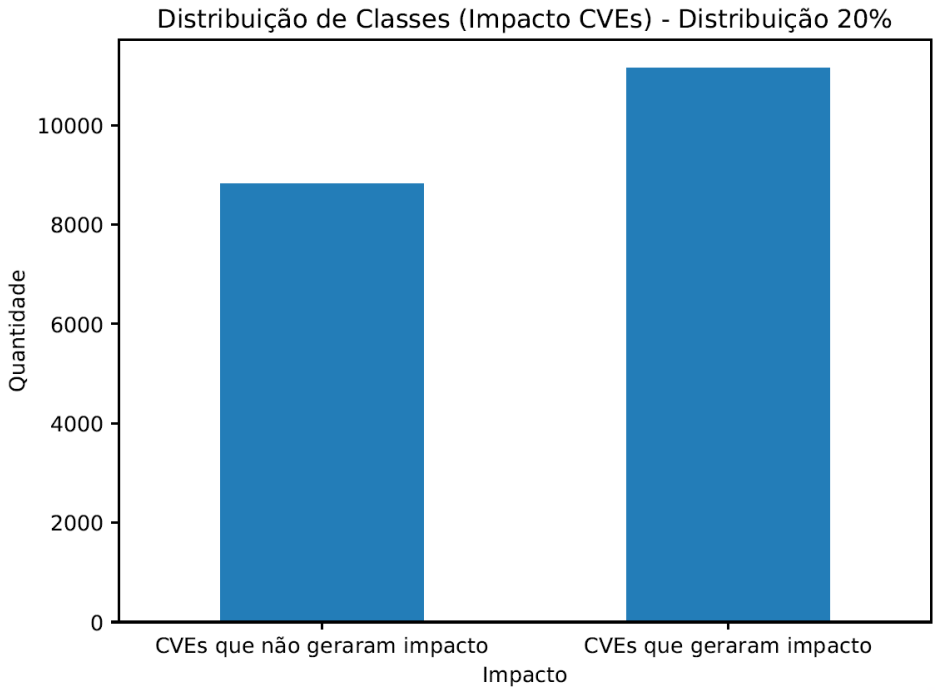
- O campo vulnerable_configuration_cpe_2_2, que possui informações a respeito da configuração do ambiente vulnerável, foi convertida de uma lista de strings para uma única string, conforme trecho de código abaixo:

```
(...)
elif d == "vulnerable_configuration_cpe_2_2":
    if type(tmp[d]) is list and len(tmp[d]) > 0:
        tmp_vc = ""
        for i in tmp[d]:
            tmp_vc = tmp_vc+";"+i
        tmp_dict[d] = tmp_vc
    else:
        tmp_dict[d] = "NotAvailable"
(...)
```

Distribuição de classes:

- Conforme saída do CSV de pré-processamento, foram criados dois gráficos com o mapa de distribuição de classes com base no campo impact:





Treinamentos, Testes e Resultados

- Após o prévio processamento do dataset “circl-cve-search-expanded.json” – escolha das informações de interesse e divisão dos dados em dois grupos, um com 80% e o outro com 20% dos dados – foi realizado o treinamento do dataset com a porção de 80% das informações nos modelos RandomForest, Kneighborn e Support-vector machine (SVM).
- Biblioteca de aprendizado de máquina scikit-learn 0.24.1 e a biblioteca de criação de gráficos e visualizações de dados Matplotlib 3.3.4, ambas para a linguagem de programação Python.
- O treinamento, o teste e a obtenção dos resultados foi realizado através do Script Python ImplementacaoModelos.py. O referido Script irá treinar, testar e gerar os resultados dos 3 modelos

Treinamentos, Testes e Resultados

- Para o tratamento das características textuais foram utilizadas os métodos TDF-IDF e Word2Vec;
- Além disso, antes de iniciar as chamadas do processamento do modelos é realizado a normalização dos dados;

Treinamentos, Testes e Resultados

- Referências:

*https://github.com/fabriciojoc/ml-cybersecuritiy-course/

*https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

*https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

*https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Treinamentos, Testes e Resultados

- RandomForest

```
def generate_models():
(...)
# ****************************** RandomForestClassifier
execute_model(RandomForestClassifier(n_estimators=100),
train_features_norm, train_label, test_features_norm, test_label,
model_name="RandomForestClassifier")
execute_kfold(RandomForestClassifier(n_estimators=100),
train_features_norm, train_label, cv,
model_name="RandomForestClassifier-KFold")
(...)
```

Treinamentos, Testes e Resultados

```
(...)
from sklearn.metrics import confusion_matrix, precision_score, mean_absolute_error
(...)

def execute_model(model, train_features_norm, train_label, test_features_norm, test_label, model_name=""):
    global generate_roc_curve
    """
    Executa um modelo conforme parâmetros
    """
    if model_name == "RandomForestClassifier" or model_name == "KNeighborsClassifier" or model_name == "SVM":
        clf = model
        clf.fit(train_features_norm, train_label)
        test_pred = clf.predict(test_features_norm)
        print(f"---------*--------- Split percentage ({model_name}) ---------*---------")
        print(f"Precisão: ", end="")
        print(precision_score(test_label, test_pred))
        print(f"Erro (mean_absolute_error): ", end="")
        print(mean_absolute_error(test_label, test_pred))
        print(f"Matriz de confusão: ")
        print(confusion_matrix(test_label, test_pred))

        if generate_roc_curve is True:
            plot_roc_curve(clf, test_features_norm, test_label)
            plt.show()
```

Treinamentos, Testes e Resultados

```
def execute_kfold(model, X, Y, cv, model_name=""):
    """
    Execução k-fold cross validation, k=cv
    """
    global generate_roc_curve

    """
    https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html
    https://scikit-learn.org/stable/glossary.html#term-random_state
    """
    kf = StratifiedKFold(n_splits=cv, random_state=None)

    count = 1
    ax = plt.gca()

    print(f"---------*--------- Kfold ({model_name}) ---------*---------")
    for train_index, test_index in kf.split(X, Y):
        X_train, X_test = X[train_index], X[test_index]
        Y_train, Y_test = Y[train_index], Y[test_index]
        clf = model
        clf.fit(X_train, Y_train)
        pred_t = clf.predict(X_test)
        print(f"Precisão: ", end="")
        print(precision_score(Y_test, pred_t))
        print(f"Erro (mean_absolute_error): ", end="")
        print(mean_absolute_error(Y_test, pred_t))
        print(f"Matriz de confusão: ")
        print(confusion_matrix(Y_test, pred_t))

        if generate_roc_curve is True:
            plot_roc_curve(clf, X_test, Y_test, ax=ax, label=f"{model_name}-{count}")
            count += 1

    if generate_roc_curve is True:
        plt.show()
```

Treinamentos, Testes e Resultados

- RandomForest
  - Resultado Split:

Treinamentos, Testes e Resultados

- RandomForest
  - Resultado K-Fold:

Treinamentos, Testes e Resultados

- RandomForest
  - Curva ROC:

Treinamentos, Testes e Resultados

- RandomForest:
  - Resultados:

Treinamentos, Testes e Resultados

- KneighborsClassifier:

```
def generate_models():
(...)
# ****************************** "KNeighborsClassifier
execute_model(KNeighborsClassifier(n_neighbors=5),
train_features_norm, train_label, test_features_norm, test_label,
model_name="KNeighborsClassifier")
execute_kfold(KNeighborsClassifier(n_neighbors=5),
train_features_norm, train_label, cv,
model_name="KNeighborsClassifier-KFold")
(...)
```

Treinamentos, Testes e Resultados

- KneighborsClassifier
  - Resultado Split:

Treinamentos, Testes e Resultados

- KneighborsClassifier
  - Resultado K-Fold:

Treinamentos, Testes e Resultados

- KneighborsClassifier
  - Curva ROC:

Treinamentos, Testes e Resultados

- KneighborsClassifier:
  - Resultados:

Treinamentos, Testes e Resultados

- SVM:

```
def generate_models():
(...)
# ****************************** SVM
execute_model(SVC(kernel="linear"), train_features_norm, train_label, test_features_norm, test_label, model_name="SVM")
execute_kfold(SVC(kernel="linear"), train_features_norm, train_label, cv, model_name="SVM-KFold")
(...)
```

Treinamentos, Testes e Resultados

- SVM:
  - Resultado Split:

Treinamentos, Testes e Resultados

- SVM:
  - Resultado K-Fold:

Treinamentos, Testes e Resultados

- SVM:
  - Curva ROC:

Treinamentos, Testes e Resultados

- SVM:
  - Resultados:

OBRIGADO.