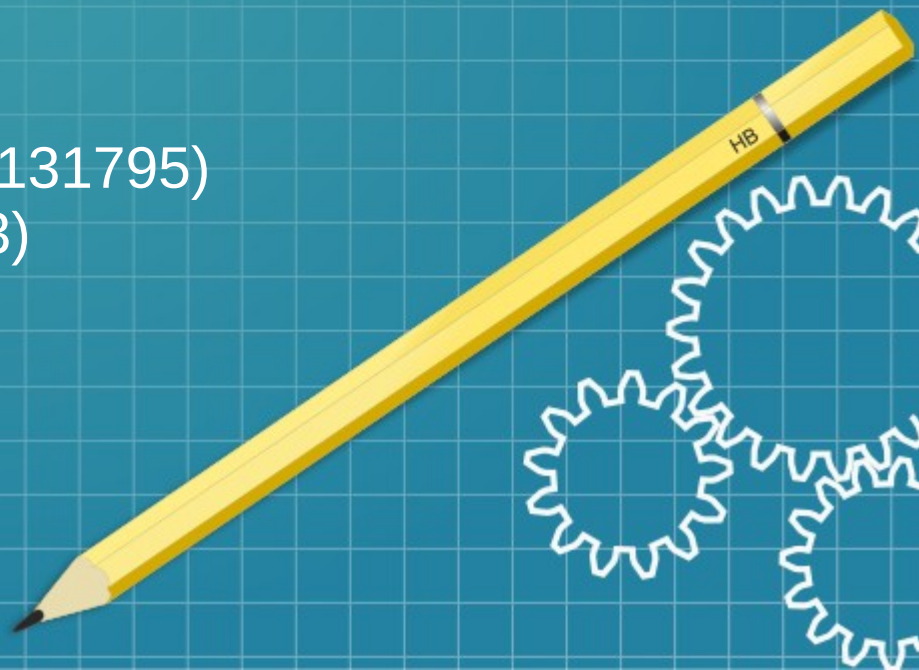
Ciência de Dados para Segurança DInf/UFPR

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"]
(...)
```

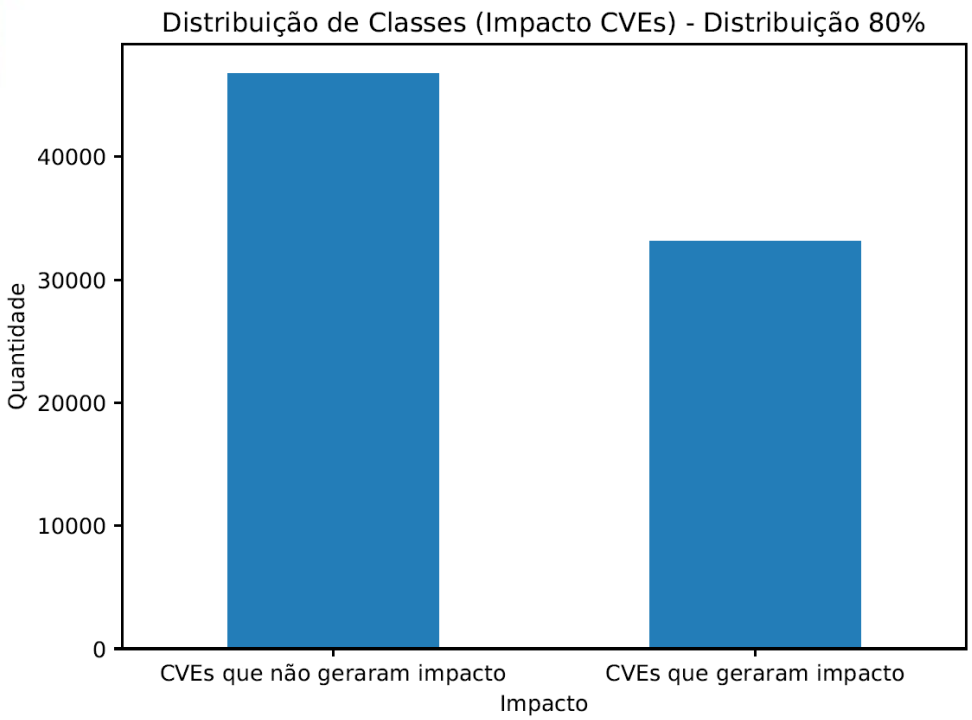
Pré-processamento

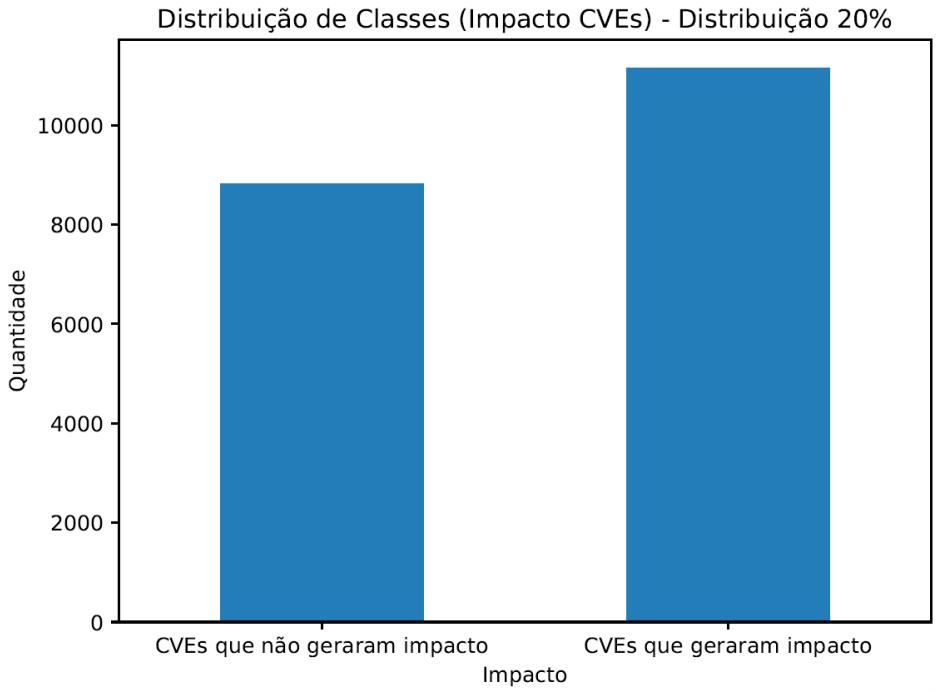
- O campo vulnerable_configuration_cpe_2_2, que possui informações a respeito da configuração do ambiente vulnerável, foi convertida de uma lista de strings para uma única string, conforme trecho de código abaixo:

```
(...)
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 (80%)

```
def generate_models(random_florest=False, k_neighbors=False, svm=False):
(...)
    # ****************************** RandomForestClassifier
    if random_florest is True:
            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":
        print(f"---------*--------- Split percentage ({model_name}) ---------*---------")
        clf = model
        clf.fit(train_features_norm, train_label)
        # Salvo o modelo treinado
        dump(clf, output_model_split[model_name])
        # Predição
        test_pred = clf.predict(test_features_norm)
        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()
    idx = 0

    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)
        dump(clf, output_model_kfold[model_name][idx])
        idx += 1
        pred_t = clf.predict(X_test)
        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 (80%):

Split percentage (RandomForestClassifier)		
Precisão	**Erro (mean_absolute_error)**	**Matriz de confusão:**
0.930338389731622	0.109060192716806 4	[6266 597] [1146 7973]

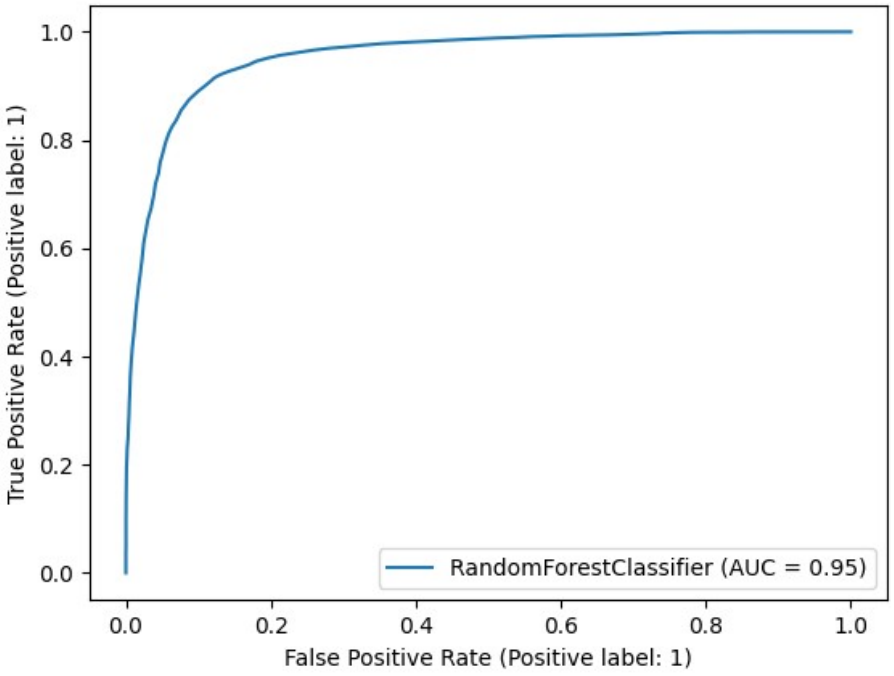
Treinamentos, Testes e Resultados

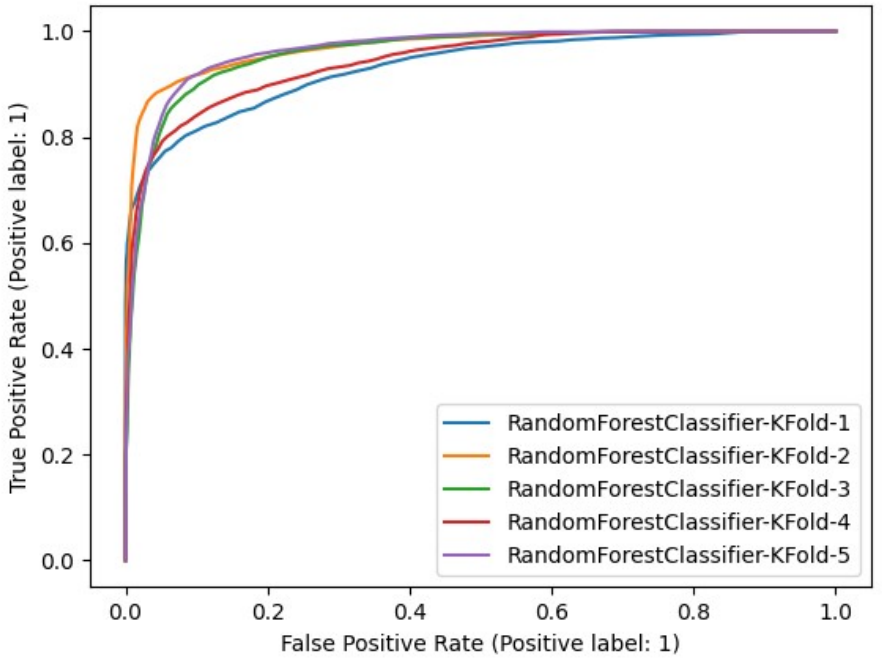
- RandomForest
  - Resultado K-Fold (80%):

K-Folds cross-validator (RandomForestClassifier)			
K-Fold	**Precisão**	**Erro (mean_absolute_error)**	**Matriz de confusão:**
1	0.9697966507177034	0.12959486938839357	[7886 101] [1556 3243]
2	0.9623915139826422	0.07524442706296441	[7831 156] [806 3992]
3	0.8889611797874648	0.09479859210011732	[7474 512] [700 4099]
4	0.7716535433070866	0.14329292139225655	[6739 1247] [585 4214]
5	0.8881856540084389	0.08752444270629645	[7456 530] [589 4210]

Treinamentos, Testes e Resultados

- RandomForest
  - Curva ROC (80%):





Treinamentos, Testes e Resultados

- RandomForest (80%):
  - Resultados Split:
    - Precisão: "0.930338389731622".
    - Média absoluta de erros: "0.109060192716806 4".
  - Resultados K-Fold:
    - Precisão mínima: "0.771653543307086 6".
    - Precisão máxima: "0.969796650717703 4".
    - Média absoluta de erros mínima: "0.0752444270629644 1".
    - Média absoluta de erros máxima: "0.143292921392256 55".

Treinamentos, Testes e Resultados

- RandomForest (20%)

```
def execute_models_production(random_florest=False, k_neighbors=False, svm=False):
(...)
    # ****************************** RandomForestClassifier
    if random_florest is True:
            execute_model_production(RandomForestClassifier(n_estimators=100),
                                     train_features_norm, train_label, test_features_norm, test_label,
                                     model_name="RandomForestClassifier")
            execute_kfold_production(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_production(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":
        print(f"---------*--------- Split percentage ({model_name})-Production---------*---------")
        # Carrega o modelo salvo em disco
        clf = load(output_model_split[model_name])
        # Predição
        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_production(test_features_norm, test_label, cv, model_name=""):
    """
    Execução k-fold cross validation de um modelo já treinado
    """
    global generate_roc_curve

    count = 1
    ax = plt.gca()

    print(f"---------*--------- Kfold ({model_name})-Production ---------*---------")
    for i in range(cv):
        # Carrega o modelo treinado
        clf = load(output_model_kfold[model_name][i])
        pred_t = clf.predict(test_features_norm)

        print(f"Precisão: ", end="")
        print(precision_score(test_label, pred_t))
        print(f"Erro (mean_absolute_error): ", end="")
        print(mean_absolute_error(test_label, pred_t))
        print(f"Matriz de confusão: ")
        print(confusion_matrix(test_label, pred_t))

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

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

Treinamentos, Testes e Resultados

- RandomForest
  - Resultado Split (20%):

Split percentage (RandomForestClassifier)		
Precisão	**Erro (mean_absolute_error)**	**Matriz de confusão:**
0.9266188959660298	0.2366221543274767	[8267 553] [4174 6983]

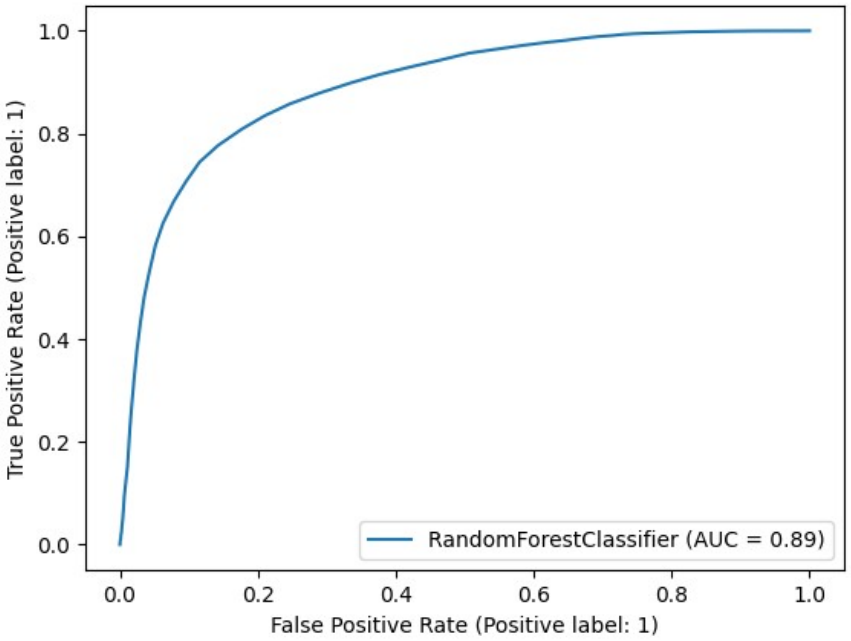
Treinamentos, Testes e Resultados

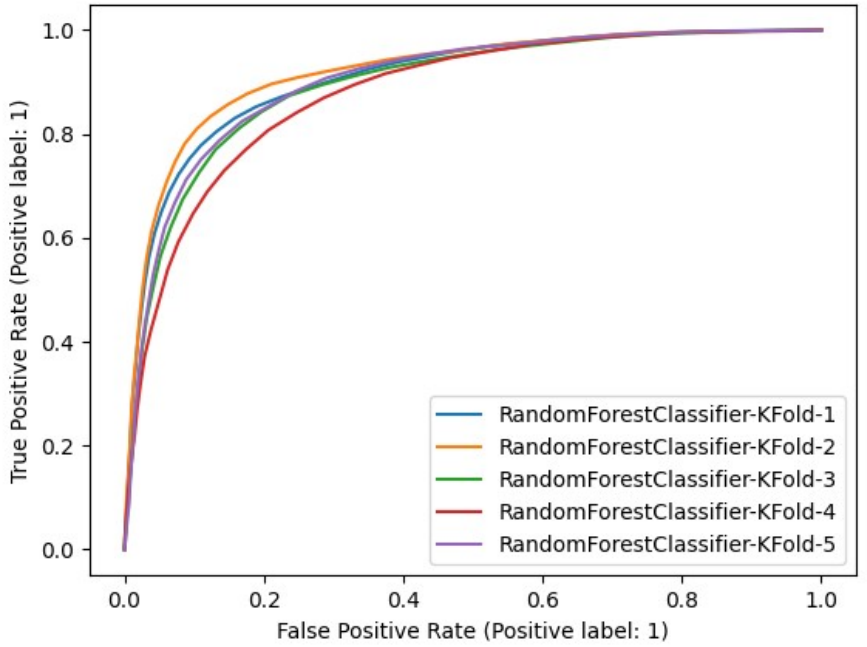
- RandomForest
  - Resultado K-Fold (20%):

K-Folds cross-validator (RandomForestClassifier)			
K-Fold	**Precisão**	**Erro (mean_absolute_error)**	**Matriz de confusão:**
1	0.9313820743259655	0.20288331581318517	[8255 565] [3488 7669]
2	0.9448778616191329	0.21024177804475147	[8389 431] [3769 7388]
3	0.9388465362943321	0.29338739550483056	[8451 369] [5492 5665]
4	0.9165649786455156	0.28507783951544274	[8273 547] [5148 6009]
5	0.9369237523781648	0.25959853831906693	[8389 431] [4755 6402]

Treinamentos, Testes e Resultados

- RandomForest
  - Curva ROC (20%):





Treinamentos, Testes e Resultados

- RandomForest (20%):
  - Resultados Split:
    - Precisão: "0.926618895966029 8".
    - Média absoluta de erros: "0.23662211543274767".
  - Resultados K-Fold:
    - Precisão mínima: "0.9165649786455156".
    - Precisão máxima: "0.944877861619132 9".
    - Média absoluta de erros mínima: "0.20288331581318517".
    - Média absoluta de erros máxima: "0.29338739550483056".

Treinamentos, Testes e Resultados

- KneighborsClassifier (80%):

```
def generate_models():
(...)
# ****************************** "KNeighborsClassifier
    if k_neighbors is True:
            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 (80%):

Split percentage (KNeighborsClassifier)		
Precisão	**Erro (mean_absolute_error)**	**Matriz de confusão:**
0.786201317462151 8	0.2606682517832561 4	[5013 1850] [2316 6803]

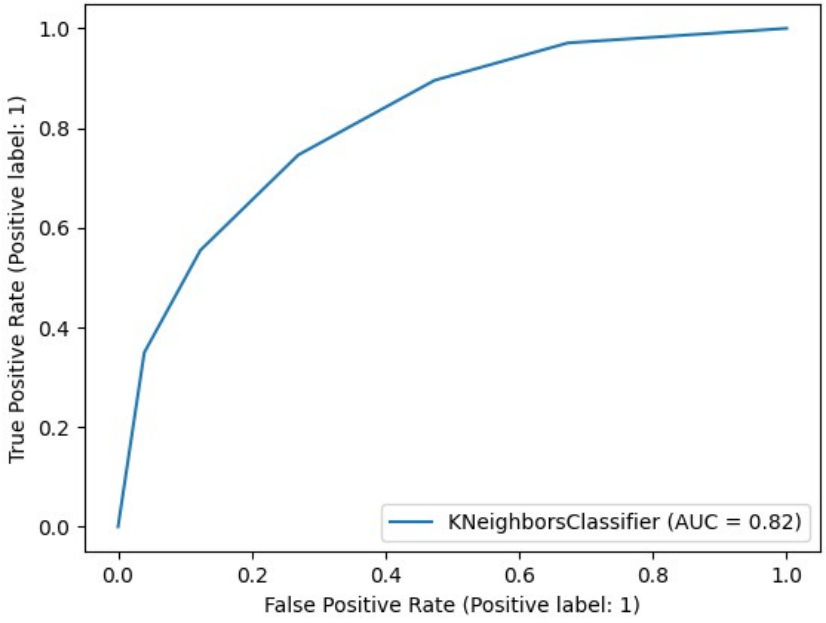
Treinamentos, Testes e Resultados

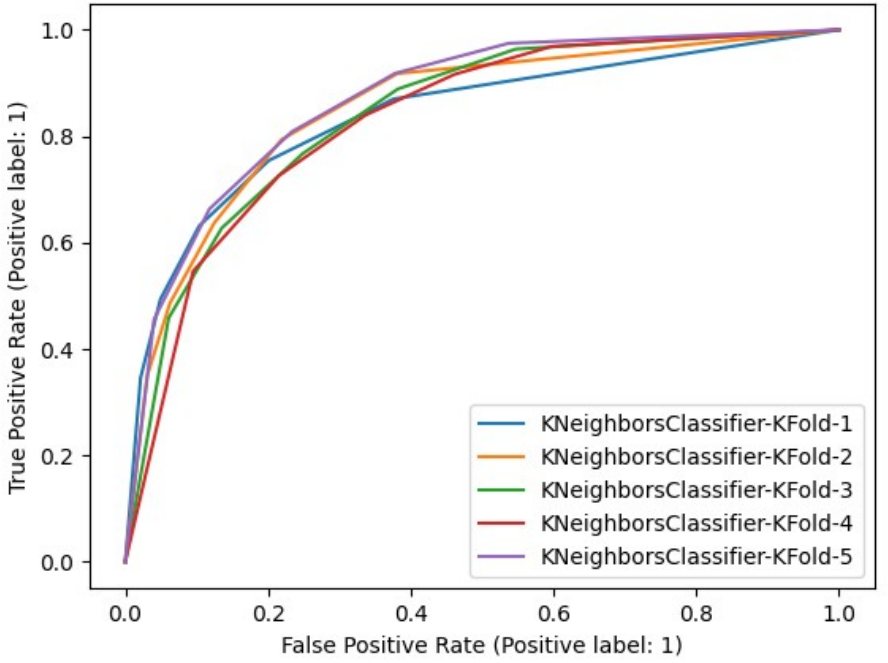
- KneighborsClassifier
  - Resultado K-Fold (80%):

K-Folds cross-validator (KNeighborsClassifier)			
K-Fold	**Precisão**	**Erro (mean_absolute_error)**	**Matriz de confusão:**
1	0.7855846512833808	0.20303456905990927	[7160 827] [1769 3030]
2	0.7542561065877128	0.2140789988267501	[6991 996] [1741 3057]
3	0.6498940677966102	0.24254986312084473	[6003 1983] [1118 3681]
4	0.5999106611078022	0.2703949941337505	[5299 2687] [770 4029]
5	0.6753698868581375	0.21775518185373485	[6121 1865] [919 3880]

Treinamentos, Testes e Resultados

- KneighborsClassifier
  - Curva ROC (80%):





Treinamentos, Testes e Resultados

- KneighborsClassifier (80%):
  - Resultados Split:
    - Precisão: "0.7862013174621518".
    - Média absoluta de erros: "0.2606682517832561 4".
  - Resultados K-Fold:
    - Precisão mínima: "0.5999106611078022".
    - Precisão máxima: "0.7855846512833808".
    - Média absoluta de erros mínima: "0.2030345690599092 7".
    - Média absoluta de erros máxima: "0.2703949941337505".

Treinamentos, Testes e Resultados

- KneighborsClassifier (20%)

```
def execute_models_production(random_florest=False, k_neighbors=False,
svm=False):
(...)
    # ****************************** KNeighborsClassifier
if k_neighbors is True:
        execute_model_production(test_features_norm, test_label,
                                                model_name="KNeighborsClassifier")
        execute_kfold_production(test_features_norm, test_label, cv,
                                                 model_name="KNeighborsClassifier-KFold")
(...)
```

Treinamentos, Testes e Resultados

- KneighborsClassifier
  - Resultado Split (20%):

Split percentage (KNeighborsClassifier)		
Precisão	**Erro (mean_absolute_error)**	**Matriz de confusão:**
0.730011348395749 5	0.335285578415177 5	[6203 2617] [4081 7076]

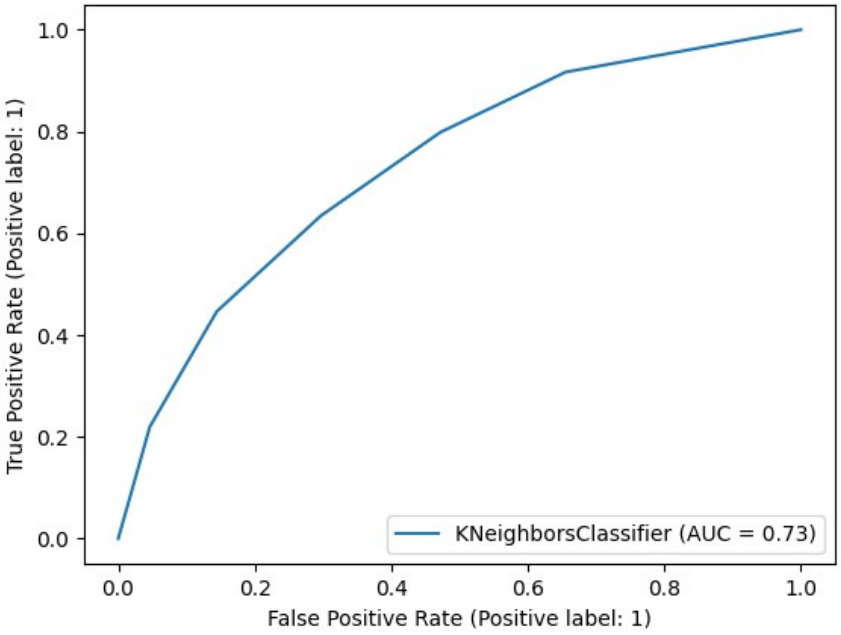
Treinamentos, Testes e Resultados

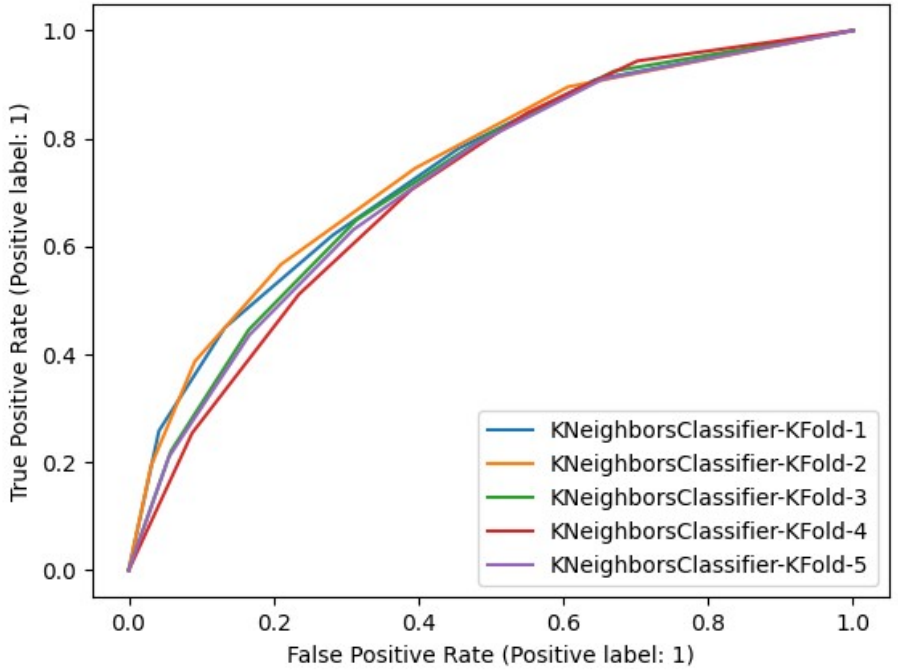
- KneighborsClassifier
  - Resultado K-Fold (20%):

K-Folds cross-validator (KNeighborsClassifier)			
K-Fold	**Precisão**	**Erro (mean_absolute_error)**	**Matriz de confusão:**
1	0.735875106202209	0.3361365570405967	[6333 2487] [4228 6929]
2	0.7730773933243673	0.3348851178855684	[6964 1856] [4834 6323]
3	0.7230492556698971	0.3349852330179707	[6048 2772] [3920 7237]
4	0.6956560205255242	0.3370876507984182	[5380 3440] [3294 7863]
5	0.7201473447252634	0.34309455874255396	[6085 2735] [4119 7038]

Treinamentos, Testes e Resultados

- KneighborsClassifier
  - Curva ROC (20%):





Treinamentos, Testes e Resultados

- KneighborsClassifier (20%):
  - Resultados Split:
    - Precisão: "0.930338389731622".
    - Média absoluta de erros: "0.109060192716806 4".
  - Resultados K-Fold:
    - Precisão mínima: "0.771653543307086 6".
    - Precisão máxima: "0.969796650717703 4".
    - Média absoluta de erros mínima: "0.075244427062964 41".
    - Média absoluta de erros máxima: "0.143292921392256 55".

Treinamentos, Testes e Resultados

- SVM (80%):

```
def generate_models(random_florest=False, k_neighbors=False,
svm=False):
(...)
# ****************************** SVM
if svm is True:
        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 (80%):

Split percentage (SVM)		
Precisão	**Erro (mean_absolute_error)**	**Matriz de confusão:**
0.894810508182601 2	0.11168814916781379	[5886 977] [808 8311]

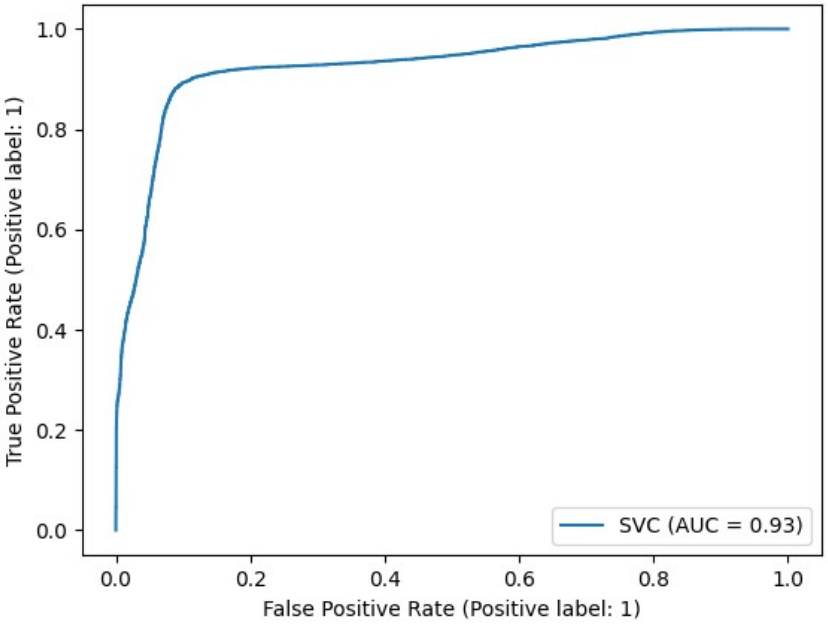
Treinamentos, Testes e Resultados

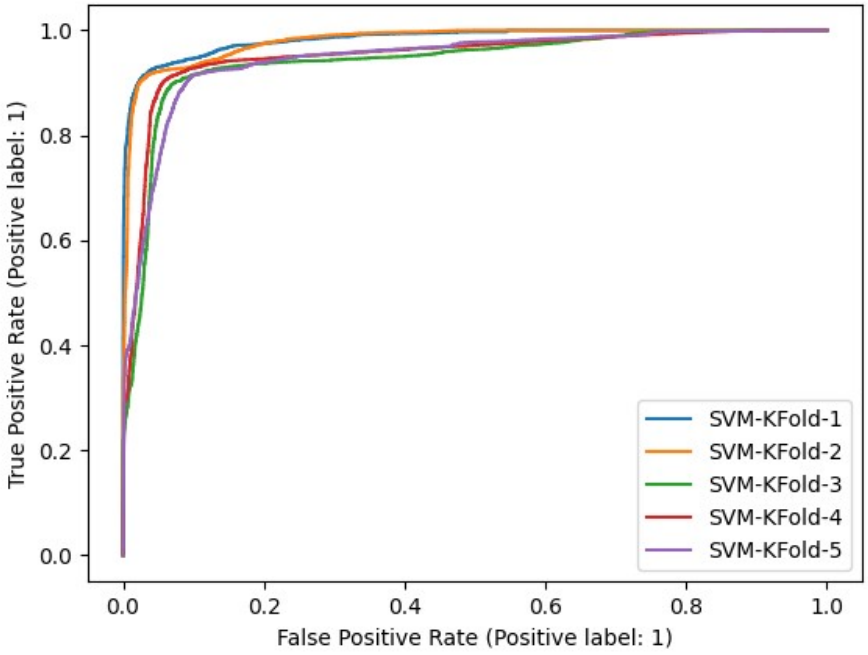
- SVM
  - Resultado K-Fold (80%):

K-Folds cross-validator (SVM)			
K-Fold	**Precisão**	**Erro (mean_absolute_error)**	**Matriz de confusão:**
1	0.9681166021407425	0.05380885343344283	[7847 140] [548 4251]
2	0.9477806788511749	0.05334376222135315	[7747 240] [442 4356]
3	0.8196114708603145	0.10512319123973406	[7011 975] [369 4430]
4	0.8542626728110599	0.08674227610481032	[7227 759] [350 4449]
5	0.809541217327728	0.11044192412983965	[6944 1042] [370 4429]

Treinamentos, Testes e Resultados

- SVM
  - Curva ROC (80%):





Treinamentos, Testes e Resultados

- SVM (80%)
  - Resultados Split:
    - Precisão: "0.894810508182601 2".
    - Média absoluta de erros: "0.1116881491678137 9".
  - Resultados K-Fold:
    - Precisão mínima: "0.809541217327728".
    - Precisão máxima: "0.968116602140742 5".
    - Média absoluta de erros mínima: "0.0533437622213531 5".
    - Média absoluta de erros máxima: "0.1051231912397340 6".

Treinamentos, Testes e Resultados

- SVM (20%)

```
def execute_models_production(random_florest=False, k_neighbors=False,
svm=False):
(...)
    # ****************************** SVM
if svm is True:
        execute_model_production(test_features_norm, test_label,
                                                model_name="SVM")
        execute_kfold_production(test_features_norm, test_label, cv,
                                                model_name="SVM-KFold")
(...)
```

Treinamentos, Testes e Resultados

- SVM
  - Resultado Split (20%):

Split percentage (SVM)		
Precisão	**Erro (mean_absolute_error)**	**Matriz de confusão:**
0.956467075278956 5	0.267707864043650 2	[8543 277] [5071 6086]

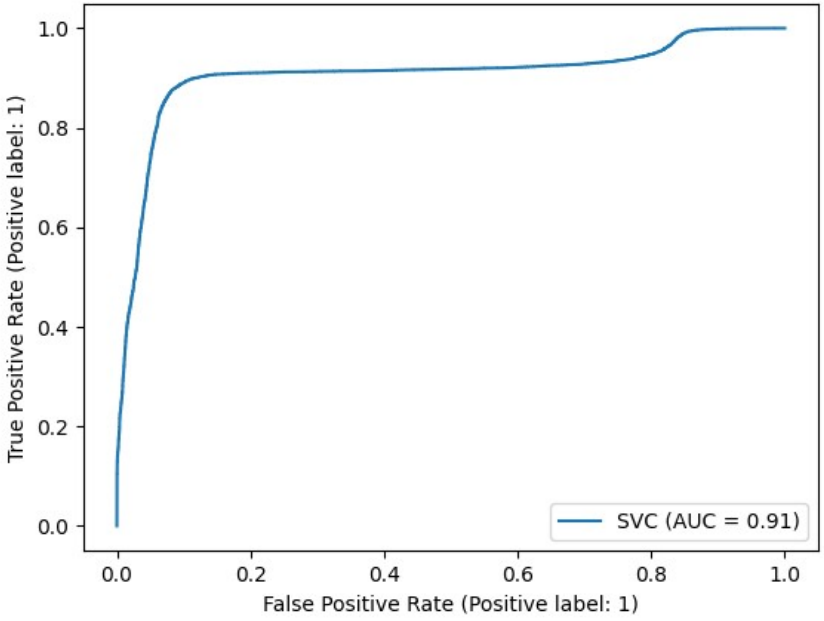
Treinamentos, Testes e Resultados

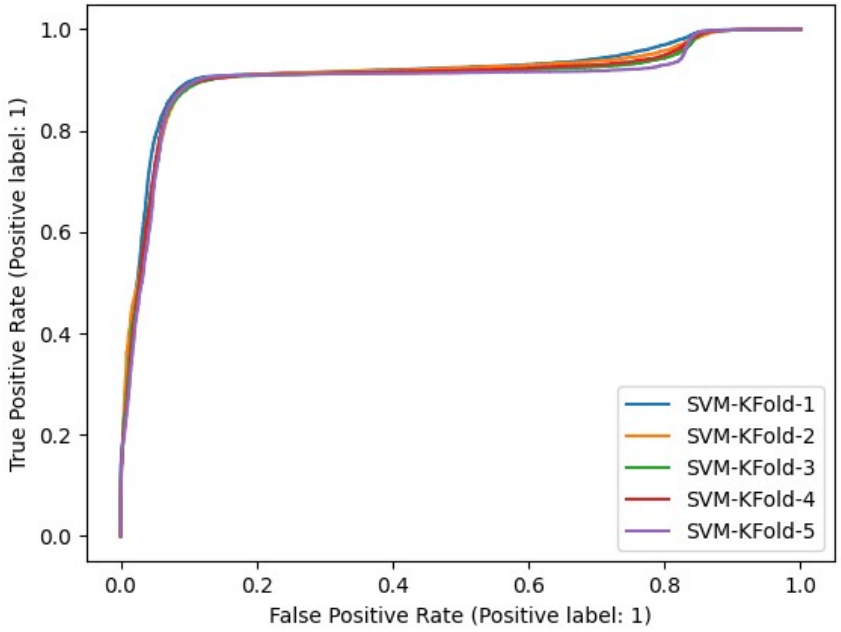
- SVM
  - Resultado K-Fold (20%):

K-Folds cross-validator (SVM)			
K-Fold	**Precisão**	**Erro (mean_absolute_error)**	**Matriz de confusão:**
1	0.9622833843017329	0.28607899083946536	[8598 222] [5493 5664]
2	0.957675402897607	0.2770185713570606	[8560 260] [5274 5883]
3	0.9607026439695763	0.3037993692746659	[8603 217] [5852 5305]
4	0.9544385655496767	0.24893627671822596	[8510 310] [4663 6494]
5	0.9482783254750679	0.244381038193923	[8458 362] [4520 6637]]

Treinamentos, Testes e Resultados

- SVM
  - Curva ROC (20%):





Treinamentos, Testes e Resultados

- SVM (20%)
  - Resultados Split:
    - Precisão: "0.956467075278956 5".
    - Média absoluta de erros: "0.267707864043650 2".
  - Resultados K-Fold:
    - Precisão mínima: "0.948278325475067 9".
    - Precisão máxima: "0.962283384301732 9".
    - Média absoluta de erros mínima: "0.244381038193923".
    - Média absoluta de erros máxima: "0.303799369274665 9".

Resultado FINAL

- 1) Com base nos resultados de precisão de margem de erros, o resultado foi satisfatório, pois mesmo que obtivemos taxas altas de precisão, a matriz de confusão nos trouxe valores altos para FN e FP.
- 2) Através das implementações e dos resultado, é possível afirmar que caso fosse implantado em ambiente de produção, seria possível obter resultado moderado na detecção de CVEs que geram impactos e dos que não gerariam impacto.
- 3) Além disso, não foram realizados ajustes de parâmetros padrão dos modelos, com excessão do modelo SVM que foi configurado para utilizar o modelo de kernel linear.
- 4) Por sim, os resultados foram neutros e provavelmente existe a possibilidade de melhorias na predição e no tempo de execução dos modelos, visto que não foram realizados trocas de parâmetros padrões além dos que já foram mencionados.

OBRIGADO.