Documentação da recriação do experimento do artigo: A multi-tiered feature selection model for android malware detection based on Feature discrimination and Information Gain

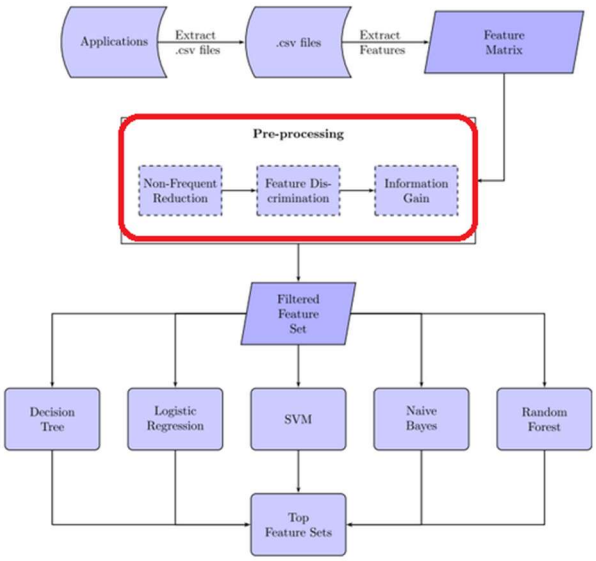
Sávio Silas Pessoa Saboia – Universidade Federal do Amazonas
savio.saboia@icomp.ufam.edu.br

1. Sobre o experimento

O experimento está focado na redução de recursos. Tal redução é seguida de classificação de malware usando algoritmos de aprendizado de máquina bem definidos. Porém, nesta recriação, o foco foi apenas na parte de pré-processamento dos dados, assim como a sua redução.

A técnica de redução de recursos remove os recursos que são muito infrequentes para ter qualquer impacto na classificação. Em segundo lugar, a abordagem de discriminação de recursos reduz o número de conjuntos de recursos. Além disso, o Ganho de Informação é aplicado aos recursos reduzidos para obter um conjunto de vinte e um recursos.

O conjunto de dados para a redução já vem em formato *.csv* (*dataset.csv*)



2. Sobre o Código e o conjunto de dados

O código segue o mesmo princípio da proposta do artigo, porém com algumas adaptações, que foram necessários por conta da distribuição das características no conjunto de dados (*dataset.csv)*.

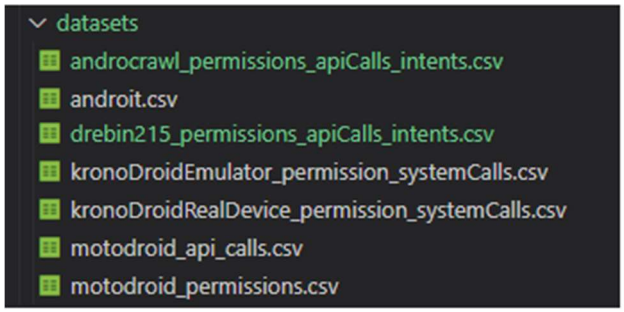


Figura 01 – Conjunto de dados

3. Experimento

Tal experimento utilizou a linguagem de programação Python 3.10.6 e testado nos ambientes Windows 10 e Windows 11.

4. Descrição do código

Nesta seção será detalhado/descrito blocos importantes do código principal do experimento.

4.1 Pré-processamento

O bloco a seguir representado na figura mostra como é feito a primeira etapa, nesta etapa é filtrada as características menos frequentes no conjunto de dados

```
15    #### Primeira etapa - Non_Frequent_Reduction
16    def Non_Frequent_Reduction(permission):
17        return len(data[data[permission]==1])/len(data)
```

Figura 02 – Primeira etapa

Aqui vemos a segunda etapa, nesta etapa é realizada a *Feature Discrimination* de cada característica. Os resultados podem ocorrer no intervalo {0,1}, e seguem a seguinte descrição:

Score(fi) = 0 {frequência igual de ocorrência em ambas as classes; sem discriminação}

Score(fi) ~ 0 {baixa frequência de ocorrência em qualquer uma das classes; pior característica discriminante}

Score(fi) ~ 1 {alta frequência de ocorrência em qualquer uma das classes; melhor característica discriminativa}

```
##### Segunda Etapa - Feature Discrimination
# fib representa a frequência do recurso fi em arquivos benignos
# fim representa a frequência do recurso fi em arquivos maliciosos
def fib(feature):
   return len(B[B[feature]==1])/len(B)

def fim(feature):
   return len(M[M[feature]==1])/len(M)
def Score(feature):
  fb = fib(feature)
  fm = fim(feature)
  score = 1.0 - (min(fb,fm)/max(fb,fm))
  return score
```

Figura 03 – Segunda etapa

Na terceira etapa, é realizado o cálculo do *Information Gain* das características do conjunto de dados, a figura a seguir representa e mostra como é feito o cálculo.

```
def entropy(labels): #labels --> RÓTULOS
    entropy=0
    label_counts = Counter(labels)
    for label in label_counts:
        prob_of_label = label_counts[label] / len(labels)
        entropy -= prob_of_label * math.log2(prob_of_label)
    return entropy

#### Terceira etapa - Information Gain
def information_gain(data, split_labels):
    info_gain = entropy(data)
    for branched_subset in split_labels:
        info_gain -= len(branched_subset) * entropy(branched_subset) / len(data)
    return info_gain
```

Figura 04 – Terceira etapa

Aqui veremos a primeira adaptação feita no código do experimento, a remoção de características muito frequentes na filtragem: INTERNET e ACCES_NETWORK_STATE, sempre aparecem nos conjuntos de dados, e sempre serão selecionadas na filtragem.

```
# exclusão de permissões comuns
for i in X.columns:
    if i == 'INTERNET':
        X = X.drop(columns=['INTERNET'])
    if i == 'ACCESS_NETWORK_STATE':
        X = X.drop(columns=['ACCESS_NETWORK_STATE'])
    X = drop_irrelevant_columns(X)
```

Figura 05 – Exclusão de características

Outra adaptação feita foi na frequência das características. No artigo é feita a filtragem das características mais frequentes em 80% (0.80) do conjunto de dados, isso faz com que haja um problema na filtragem quando se é feita em *datasets* pequenos. A adaptação feita foi que a frequência seria a partir da característica mais frequente, ainda assim olharíamos pro conjunto de dados como um todo.

```
## corte a partir da característica mais frequente
ft_sum = X.sum()
ft_max = ft_sum.max()
th = ft_max * 0.80
select_ft = list()
for index, value in ft_sum.items():
    if value >= th:
        select_ft.append(index)
```

Figura 06 – Corte das características

5. Testes e resultados

   Para a realização dos testes, seguisse o seguinte padrão de comandos:

   python MTmain.py -d "dataset.csv"

   O teste a seguir foi realizado usando o programa Visual Studio Code no Windows 11

```
PS C:\Users\SVO-AVELL\CaracteristicasGerais\Adaptações\códigos> python MTmain_ad.py
-d "C:\Users\SVO-AVELL\CaracteristicasGerais\Adaptações\datasets\motodroid_permissions.csv"
```

Aqui vemos as primeiras características filtradas na primeira etapa bem como o valor de suas frequências:

```
Non-Frequent Reduction --> FREQUÊNCIA DE CADA CARACTERÍSTICA
API Call :: Landroid/widget/EdgeEffect.finish() 0.8145275337475895
API Call :: Landroid/view/ViewPropertyAnimator.setStartDelay() 0.7829012213413328
API Call :: Landroid/view/animation/Animation.setInterpolator() 0.7585386758088708
API Call :: Landroid/widget/EdgeEffect.onAbsorb() 0.8547675166059567
API Call :: Landroid/os/Handler.init() 0.9631669166488108
API Call :: Landroid/view/accessibility/AccessibilityNodeInfo.setBoundsInParent() 0.7769659310049282
API Call :: Landroid/os/Handler.removeMessages() 0.9107563745446754
API Call :: Landroid/widget/TextView.setTextSize() 0.8757231626312406
API Call :: Landroid/webkit/WebSettings.setAllowFileAccess() 0.6416755946003857
```

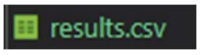
O valor das discriminações de cada característica, na segunda etapa:

```
Feature Discrimination --> SCORE
API Call :: Landroid/widget/EdgeEffect.finish() 0.4749869697339024
API Call :: Landroid/view/ViewPropertyAnimator.setStartDelay() 0.4711006426270209
API Call :: Landroid/view/animation/Animation.setInterpolator() 0.44229834946265145
API Call :: Landroid/widget/EdgeEffect.onAbsorb() 0.5013276356648708
API Call :: Landroid/os/Handler.init() 0.367360912023444
API Call :: Landroid/view/accessibility/AccessibilityNodeInfo.setBoundsInParent() 0.4801060004419677
API Call :: Landroid/os/Handler.removeMessages() 0.45359493561338127
API Call :: Landroid/widget/TextView.setTextSize() 0.47339895488742145
API Call :: Landroid/webkit/WebSettings.setAllowFileAccess() 0.3425143251658904
```

Resultado da terceira etapa:

```
Information Gain
API Call :: Landroid/widget/EdgeEffect.finish() 0.5064828960488814
API Call :: Landroid/view/ViewPropertyAnimator.setStartDelay() 0.5064828960488814
API Call :: Landroid/view/animation/Animation.setInterpolator() 0.5064828960488814
API Call :: Landroid/widget/EdgeEffect.onAbsorb() 0.5064828960488814
API Call :: Landroid/os/Handler.init() 0.5064828960488814
API Call :: Landroid/view/accessibility/AccessibilityNodeInfo.setBoundsInParent() 0.5064828960488814
API Call :: Landroid/os/Handler.removeMessages() 0.5064828960488814
API Call :: Landroid/widget/TextView.setTextSize() 0.5064828960488814
API Call :: Landroid/webkit/WebSettings.setAllowFileAccess() 0.5064828960488814
```

Neste teste, foram selecionadas 75 características do conjunto de dados motodroid_permissions.csv.

O final do experimento é gerado um outro arquivo .csv com as características selecionadas, com o nome de. 

Este arquivo servirá para a análise posterior. Aqui vemos uma análise aplicando Random Forest e SVM:

```
Random Forest Classifier
       Time  Accuracy  Precision    Recall  F1_Score       Fpr
0  1.022571  0.962288    0.72791  0.608441  0.662835  0.014755

SVM
       Time  Accuracy  Precision    Recall  F1_Score       ROC
0  2.494901  0.949075   0.611821  0.449004  0.517918  0.715261
```