Disciplina - ci1030-ERE2-CiênciaDeDados - UFPR

Projeto de Mineração de Dados

Heloise Acco Tives - Matrícula: 202000144019

Roberta Samistraro Tomigian - GRR - 20171631

Link do vídeo

Link do github

Introdução

Os dados a serem utilizados nesse projeto trata-se do conjunto de dados darknet completo cobrindo o tráfego Tor e VPN disponibilizado publicamente pelo Canadian Institute for Cybersecurity através do link: https://www.unb.ca/cic/datasets/darknet2020.html.

O dataset é formado por 85 atributos e 141.530 registros, sendo distribuídos através de 6 atributos textuais e 79 atributos numéricos, conforme apresentado na seção de pré-processamento dos dados.

Entre os objetivos desde projeto tem-se identificação, a partir da análise do dataset através dos algorimos KNN, Random Forest e a rede neural MLP, de novas formas para categorizar o tráfego de rede e identificar padrões de malwares ou atividades maliciosas.

O Dataset

O dataset utilizado apresenta conjuntos de dados oriundos de tráfego capturado regular, VPN e Tor para sete categorias de aplicativos: Navegação (Firefox e Chrome), Chat (ICQ, AIM, Skype, Facebook e Hangouts),vEmail (SMTPS, POP3S e IMAPS), Transferência de arquivos (Skype, FTP sobre SSH (SFTP) e FTP sobre SSL (FTPS) usando Filezilla e um serviço externo), Streaming (Vimeo e Youtube), VoIP (Facebook, Skype e chamadas de voz do Hangouts) e P2P (uTorrent e Transmissão (BitTorrent)). A categorização desses dados pode ser conferida no "Label2" - atributo 85 do dataset.

O atributo utilizado para guiar a execução dos algoritmos foi o "Label" - atributo 84, que se refere aos dados de identificação de tráfego TOR, Não-Tor, VPN e Não VPN. Sendo essas categorias o foco de aprofundamento deste estudo.

Pré-processamento dos dados

A atividade de pré-processmento dos dados envolveu o estudo do dataset a ser utilizado, a identificação de atributos a serem mantidos ou excluídos, como também a extração de características e adequação da base de dados de forma a possibilitar a execução do projeto.

Atributos e representação

Os rótulos dos atributos do dataset obtido são apresentados na tabela abaixo, juntamente com a identificação do tipo de dados (textual e numérico) e uma breve descrição sobre o que os dados se referem.

Id	Atributo	Tipo	Descrição
1	Flow ID	Textual	ID do fluxo
2	Src IP	Textual	IP de Origem
3	Src Port	Numérico	Porta de Origem
4	Dst IP	Numérico	IP de Destino
5	Dst Port	Textual	Porta de Destino
6	Protocol	Numérico	Protocolo
7	Timestamp	Textual	Marca temporal
8	Flow Duration	Numérico	Duração do fluxo em microssegundos
9	Total Fwd Packet	Numérico	Total de Pacotes enviados
10	Total Bwd packets	Numérico	Total de Pacotes recebidos
11	Total Length of Fwd Packet	Numérico	Tamanho total do pacote enviado
12	Total Length of Bwd Packet	Numérico	Tamanho total do pacote recebido
13	Fwd Packet Length Max	Numérico	Tamanho máximo do pacote enviado
14	Fwd Packet Length Min	Numérico	Tamanho mínimo do pacote enviado
15	Fwd Packet Length Mean	Numérico	Tamanho médio do pacote enviado
16	Fwd Packet Length Std	Numérico	Valor so desvio padrão do pacote enviado
17	Bwd Packet Length Max	Numérico	Tamanho máximo dos pacotes recebidos
18	Bwd Packet Length Min	Numérico	Tamanho mínimo dos pacotes recebidos
19	Bwd Packet Length Mean	Numérico	Tamanho médio dos pacotes recebidos
20	Bwd Packet Length Std	Numérico	Valor do desvio padrão dos pacotes recebidos
21	Flow Bytes/s	Numérico	Número de bytes do fluxo por segundo
22	Flow Packets/s	Numérico	Número de pacotes de fluxo por segundo
23	Flow IAT Mean	Numérico	Tempo médio entre dois pacotes enviados no fluxo
24	Flow IAT Std	Numérico	Valor do desvio padrão entre dois pacotes enviados no fluxo
25	Flow IAT Max	Numérico	Tempo máximo entre dois pacotes enviados no fluxo
26	Flow IAT Min	Numérico	Tempo mínimo entre dois pacotes enviados no fluxo
27	Fwd IAT Total	Numérico	Tempo total entre dois pacotes enviados enviados
28	Fwd IAT Mean	Numérico	Tempo médio entre dois pacotes enviados
29	Fwd IAT Std	Numérico	Valor do desvio padrão entre dois pacotes enviados
30	Fwd IAT Max	Numérico	Tempo máximo entre dois pacotes enviados

Id	Atributo	Tipo	Descrição
31	Fwd IAT Min	Numérico	Tempo mínimo entre dois pacotes enviados
32	Bwd IAT Total	Numérico	Tempo total entre dois pacotes recebidos
33	Bwd IAT Mean	Numérico	Tempo médio entre dois pacotes recebidos
34	Bwd IAT Std	Numérico	Valor do desvio padrão entre dois pacotes recebidos
35	Bwd IAT Max	Numérico	Tempo máximo entre dois pacotes recebidos
36	Bwd IAT Min	Numérico	Tempo mínimo entre dois pacotes recebidos
37	Fwd PSH Flags	Numérico	Número de vezes que a bandeira PSH foi definida em pacotes que foram enviados
38	Bwd PSH Flags	Numérico	Número de vezes que a bandeira PSH foi definida em pacotes que foram recebido
39	Fwd URG Flags	Numérico	Número de vezes que a bandeira URG foi definida em pacotes que foram enviados
40	Bwd URG Flags	Numérico	Número de vezes que a bandeira URG foi definida em pacotes que foram recebido
41	Fwd Header Length	Numérico	Total de bytes usados para cabeçalhos enviados
42	Bwd Header Length	Numérico	Total de bytes usados para cabeçalhos recebidos
43	Fwd Packets/s	Numérico	Número de pacotes enviados por segundo
44	Bwd Packets/s	Numérico	Número de pacotes recebidos por segundo
45	Packet Length Min	Numérico	Comprimento mínimo de um pacote
46	Packet Length Max	Numérico	Comprimento máximo de um pacote
47	Packet Length Mean	Numérico	Comprimento médio de um pacote
48	Packet Length Std	Numérico	Valoe do desvio padrão do comprimento dos pacotes
49	Packet Length Variance	Numérico	Valor da variância do comprimentos dos pacotes
50	FIN Flag Count	Numérico	Número de pacotes com FIN
51	SYN Flag Count	Numérico	Número de pacotes com SYN
52	RST Flag Count	Numérico	Número de pacotes com RST
53	PSH Flag Count	Numérico	Número de pacotes com PSH
54	ACK Flag Count	Numérico	Número de pacotes com ACK
55	URG Flag Count	Numérico	Número de pacotes com URG
56	CWE Flag Count	Numérico	Número de pacotes com CWR
57	ECE Flag Count	Numérico	Número de pacotes com ECE
58	Down/Up Ratio	Numérico	Taxa de download e upload
59	Average Packet Size	Numérico	Tamanho médio do pacote
60	Fwd Segment Size Avg	Numérico	Tamanho médio observado enviados
61	Bwd Segment Size Avg	Numérico	Taxa média do número de bytes em massa recebidos
62	Fwd Bytes/Bulk Avg	Numérico	Taxa média do número de bytes em massa enviados
63	Fwd Packet/Bulk Avg	Numérico	Taxa média do número de pacotes em massa enviados
64	Fwd Bulk Rate Avg	Numérico	Número médio de taxa em massa enviados
65	Bwd Bytes/Bulk Avg	Numérico	Taxa média do número de bytes em massa recebidos
66	Bwd Packet/Bulk Avg	Numérico	Taxa média do número de pacotes em massa recebidos
67	Bwd Bulk Rate Avg	Numérico	Número médio de taxa em massa recebidos
68	Subflow Fwd Packets	Numérico	Número médio de pacotes em um subfluxo enviados
69	Subflow Fwd Bytes	Numérico	Número médio de bytes em um subfluxo enviados
70	Subflow Bwd Packets	Numérico	Número médio de pacotes em um subfluxo recebidos
71	Subflow Bwd Bytes	Numérico	Número médio de bytes em um subfluxo recebidos
72	FWD Init Win Bytes	Numérico	Número total de bytes enviados na janela inicial enviados

Id	Atributo	Tipo	Descrição
73	Bwd Init Win Bytes	Numérico	Número total de bytes enviados na janela inicial recebidos
74	Fwd Act Data Pkts	Numérico	Contagem de pacotes com pelo menos 1 byte de carga útil de dados TCP enviado
75	Fwd Seg Size Min	Numérico	Tamanho mínimo do segmento observado enviados
76	Active Mean	Numérico	Tempo médio em que um fluxo estava ativo antes de ficar ocioso
77	Active Std	Numérico	Valoe do desvio padrão em que um fluxo estava ativo antes de ficar ocioso
78	Active Max	Numérico	Tempo máximo em que um fluxo ficou ativo antes de se tornar ocioso
79	Active Min	Numérico	Tempo mínimo em que um fluxo esteve ativo antes de se tornar ocioso
80	Idle Mean	Numérico	Tempo médio em que um fluxo ficou ocioso antes de se tornar ativo
81	Idle Std	Numérico	Valor do desvio padrão em que um fluxo estava ocioso antes de se tornar ativo
82	Idle Max	Numérico	Tempo máximo em que um fluxo ficou ocioso antes de se tornar ativo
83	Idle Min	Numérico	Tempo mínimo em que um fluxo ficou ocioso antes de se tornar ativo
84	Label	Textual	Rótulo 1
85	Label	Textual	Rótulo 2

Seleção de atributos e adequação do dataset

Após o estudo e entendimento do dataset, foram removidos os seguintes grupos de atributos:

- Atributos exclusivos dos registros. Ao todo são 3 atributos com essa característica, sendo os IDs: 1, 7 e 58.
- Atributos que se tratam de resultados estatísticos calculados a partir de atributos originais de valores mínimos e máximos, ou seja, os atributos referentes ao cálculo de desvio padrão, média e variância. Ao todo são 17 atributos com essa característica, sendo os IDs: 15, 16, 19, 20, 23, 24, 28, 29, 33, 34, 47, 48, 49, 76, 77, 80, 81.
- Atributos com todos os registros com valor 0. Ao todo são 13 atributos com essa característica, sendo os IDs: 38, 39, 40, 55, 56, 57, 62, 63, 64, 65, 70, 78, 79.
- 10 atributos considerados inconsistentesno estudo, sendo os IDs: 21, 22, 43, 44, 59, 60, 61, 68, 69 e 71

O atributo utilizado para guiar a execução dos algoritmos foi o "Label" - atributo 84, que se refere aos dados de identificação de tráfego TOR, Não-Tor, VPN e Não VPN. Sendo essas categorias o foco de aprofundamento deste estudo.

Para execução do projeto, o atributo 85 - "Label", precisou ser renomeado para Label 2.

Extração de características

A atividade de extração de características iniciou-se com a identificação dos tipos de atributos presentes no dataset, conforme pode ser verificado em detalhes na seção de "Atributos e Representação".

A extração das características através da execução das atividades de vetorização dos atributos textuais, inclusão dos atributos textuais e numéricos em uma única matriz e normalização da base criada. O código Python que utilizado para realização dessa etapa é demonstrado e explicado na seção Criação de Modelos.

Criação de Modelos

Comum a todos

```
from google.colab import files
uploaded =files.upload()
```

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving Darknet (1).CSV to Darknet (1).CSV
```

Double-click (or enter) to edit

```
import pandas as pd
# Identificação do Dataset
data_path = "Darknet (1).CSV"
# Leitura do CSV
data = pd.read_csv(data_path, keep_default_na=False)
# categorical = data.columns[(data.dtypes.values == np.dtype('float64'))]
# data[categorical] = data[categorical].astype(int)
```

```
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2718:
  interactivity=interactivity, compiler=compiler, result=result)
```

```
# Separação dos atributos numéricos
NUMERICAL_ATTRIBUTES = ['SrcPort','DstPort','Protocol','FlowDuration','TotalFwdPacl
                        'TotalLengthofBwdPacket', 'FwdPacketLengthMax','FwdPacketLe
                        'BwdPacketLengthMin','FlowIATMax','FlowIATMin','FwdIATTota
                        'BwdIATMax','BwdIATMin','BwdURGFlags','FwdHeaderLength','Bv
                        'PacketLengthMin','PacketLengthMax','FINFlagCount','SYNFlag
                        'FwdSegSizeMin']

# Separação dos atributos textuais
TEXTUAL_ATTRIBUTES = ['Label2','SrcIP','DstIP']

# Separação do rótulo usado para classificação
LABEL = 'Label'

# Separação dos atributos não usados
UNUSED_ATTRIBUTES = ['FlowID','Timestamp','FwdPacketLengthMean','FwdPacketLengthSt
                     'BwdPacketLengthStd','FlowIATMean','FlowIATStd','FwdIATMean',
                     'BwdIATStd','PacketLengthMean','PacketLengthStd','PacketLengtl
                     'FlowBytess', 'SubflowFwdPackets','SubflowFwdBytes','SubflowBv
                     'ActiveMean','ActiveStd','IdleMean','FlowPacketss','IdleStd',
```

```
                      'FwdURGFlags','URGFlagCount','CWEFlagCount','ECEFlagCount','F
                      'FwdPacketBulkAvg','FwdBulkRateAvg','BwdBytesBulkAvg','Active
                      'BwdPacketss','AveragePacketSize','FwdSegmentSizeAvg','BwdSeg
```

```
label = data[LABEL].values
# Remoção dos atributos e rótulos não utilizados.
for a in UNUSED_ATTRIBUTES:
    del data[a]
del data[LABEL]
```

```
#Verificação do tipo dos atributos para conferência dos tipos dos dados.
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 141530 entries, 0 to 141529
Data columns (total 39 columns):
 #   Column                  Non-Null Count   Dtype 
---  ------                  --------------   ----- 
 0   SrcIP                   141530 non-null  object
 1   SrcPort                 141530 non-null  int64 
 2   DstIP                   141530 non-null  object
 3   DstPort                 141530 non-null  int64 
 4   Protocol                141530 non-null  int64 
 5   FlowDuration            141530 non-null  int64 
 6   TotalFwdPacket          141530 non-null  int64 
 7   TotalBwdpackets         141530 non-null  int64 
 8   TotalLengthofFwdPacket  141530 non-null  int64 
 9   TotalLengthofBwdPacket  141530 non-null  int64 
 10  FwdPacketLengthMax      141530 non-null  int64 
 11  FwdPacketLengthMin      141530 non-null  int64 
 12  BwdPacketLengthMax      141530 non-null  int64 
 13  BwdPacketLengthMin      141530 non-null  int64 
 14  FlowIATMax              141530 non-null  int64 
 15  FlowIATMin              141530 non-null  int64 
 16  FwdIATTotal             141530 non-null  int64 
 17  FwdIATMax               141530 non-null  int64 
 18  FwdIATMin               141530 non-null  int64 
 19  BwdIATTotal             141530 non-null  int64 
 20  BwdIATMax               141530 non-null  int64 
 21  BwdIATMin               141530 non-null  int64 
 22  BwdURGFlags             141530 non-null  int64 
 23  FwdHeaderLength         141530 non-null  int64 
 24  BwdHeaderLength         141530 non-null  int64 
 25  PacketLengthMin         141530 non-null  int64 
 26  PacketLengthMax         141530 non-null  int64 
 27  FINFlagCount            141530 non-null  int64 
 28  SYNFlagCount            141530 non-null  int64 
 29  RSTFlagCount            141530 non-null  int64 
 30  PSHFlagCount            141530 non-null  int64 
 31  ACKFlagCount            141530 non-null  int64 
 32  BwdPacketBulkAvg        141530 non-null  int64 
 33  BwdBulkRateAvg          141530 non-null  int64 
 34  FWDInitWinBytes         141530 non-null  int64 
 35  BwdInitWinBytes         141530 non-null  int64 
 36  FwdActDataPkts          141530 non-null  int64 
 37  FwdSegSizeMin           141530 non-null  int64 
 38  Label2                  141530 non-null  object
```

```
dtypes: int64(36), object(3)
memory usage: 42.1+ MB
```

```
#Depois disso, divide-se o conjunto de dados em 5 partes.
#(As 4 primeiras são para o conjunto de treinamento (80% solicitado) e a última, pa
def split_data(data):
    # Divisão dos dados em 5 partes.
    mid = int((len(data) + 1)/5)
    # Atribuição dos dados para treinamento e teste.
    # %cada classe, ver dist classe para vir todas
    train_data = data[mid:]
    test_data = data[:mid]
    # Retorno dos dados de treinamento e teste.
    return(train_data, test_data)


train_data, test_data = split_data(data)
train_label, test_label = split_data(label)
test_label_RF = test_label


#Criação de um método para extrair recursos, dados um conjunto de treinamento e tes
from sklearn.feature_extraction.text import TfidfVectorizer
#extração de features dos atributos textuais
def textual_feature_extraction(train_data, test_data, extractor=TfidfVectorizer(max
    vectorizer = extractor
    # treinamento
    vectorizer.fit(train_data)
    # transforma treino e teste em features
    train_features = vectorizer.transform(train_data)
    test_features = vectorizer.transform(test_data)
    return(train_features, test_features)


#Obtenção de atributos numéricos:
train_features = train_data[NUMERICAL_ATTRIBUTES].values
test_features = test_data[NUMERICAL_ATTRIBUTES].values


#Verificação do tamanho das bases:
train_features.shape, test_features.shape
```

```
((113224, 36), (28306, 36))
```

```
#Obtenção de atributos textuais e união com a matriz de recursos já inicializada:
import numpy as np
# extração das features de cada atributo textual
for a in TEXTUAL_ATTRIBUTES:
    train_texts, test_texts = textual_feature_extraction(train_data[a], test_data[a
    train_features = np.concatenate((train_features, train_texts.toarray()), axis=1
    test_features = np.concatenate((test_features, test_texts.toarray()), axis=1)


#Verificação do tamanho das bases:
train_features.shape, test_features.shape
```

```
((113224, 246), (28306, 246))
```

```
#Criação de método de normalização, que normaliza o conjunto de treinamento e teste
from sklearn.preprocessing import MinMaxScaler
def normalization(train_features, test_features, scaler=MinMaxScaler()):
    # treinamento
    scaler.fit(train_features)
    # transforma features normalizadas
    train_features_norm = scaler.transform(train_features)
    test_features_norm = scaler.transform(test_features)
    return(train_features_norm, test_features_norm)


train_features_norm, test_features_norm = normalization(train_features, test_featu


#A partir de agora, tem-se um conjunto de treinamento e um conjunto de teste pronto
train_features_norm.shape, test_features_norm.shape
```

```
((113224, 246), (28306, 246))
```

```
#Gráfico para demonstração do tamanho das classes.
import matplotlib.pyplot as plt
import numpy as np

classes = ['Classe de treinamento','Classe de teste']

data1 = len(train_features_norm)
data2 = len(test_features_norm)
df = [data1,data2]

porc1 = data1 / data2 * 20
porc2 = 100- porc1


print("Quantidade de Registros Classe de Treinamento:", data1, "-", porc1, "%",  "

plt.bar(classes, df, color="grey")


plt.show()
```

```
Quantidade de Registros Classe de Treinamento: 113224 - 80.0 %
Quantidade de Registros Classe de Teste: 28306 - 20.0 %
```


```
# Cálculo média de erro absoluto
from sklearn.metrics import mean_absolute_error

def fn_mean_absolute_error(label, pred):
  dic_label = {'Non-Tor': 1.0, 'NonVPN': 2.0, 'Tor': 3.0, 'VPN': 4.0}
  label_number = [dic_label[i] for i in label]
  pred_number = [dic_label[i] for i in pred]
  print(mean_absolute_error(label_number,pred_number))
```

```
# Cálculo matriz de confusão
from sklearn.metrics import confusion_matrix

def fn_cunfusion_matrix(train_label, train_pred):
  print(confusion_matrix(train_label,train_pred))


# Cálculo da precisão
from sklearn.metrics import precision_score

def fn_precision_score(label,pred):
  print(precision_score(label,pred,average='micro')) ##'micro' ou None


# Cálculo da curva roc
import numpy as np
from sklearn import metrics
from sklearn import preprocessing
from sklearn.metrics import roc_curve

def fn_roc_curve(label, pred):
  dic_label = {'Non-Tor': 1.0, 'NonVPN': 2.0, 'Tor': 3.0, 'VPN': 4.0}
  label_number = [dic_label[i] for i in label]
  pred_number = [dic_label[i] for i in pred]
  fpr, tpr, _ = roc_curve(label_number,pred_number,pos_label=2)
  return fpr, tpr



# Gráfico da curva roc
import matplotlib.pyplot as plt
from sklearn import datasets, metrics, model_selection, svm

def fn_roc_curve_plot():
```

```
def fn_roc_curve_plot():
  train_label_roc,train_pred_roc = datasets.make_classification(random_state=0)
  X_train, X_test, y_train, y_test = model_selection.train_test_split(train_label_
  clf = svm.SVC(random_state=0)
  clf.fit(X_train, y_train)
  metrics.plot_roc_curve(clf, X_train, y_train)
  plt.show()


# Cálculo kfold
import numpy as np
from sklearn.model_selection import KFold
from sklearn import metrics, model_selection
def fn_KFold(classifier, train_features_norm, train_label):
  kf = KFold(n_splits=5)
  i = 1
  for train_index, test_index in kf.split(train_features_norm):
    x_train, x_test = train_features_norm[train_index], train_features_norm[test_i
    y_train, y_test = train_label[train_index], train_label[test_index]

    classifier.fit(x_train, y_train)
    y_pred = classifier.predict(x_test)
    dic_label = {'Non-Tor': 1.0, 'NonVPN': 2.0, 'Tor': 3.0, 'VPN': 4.0}
    label_number = [dic_label[i] for i in y_test]
    pred_number = [dic_label[i] for i in y_pred]
    print('kfold[', i, ']:')
    fn_precision_score(label_number, pred_number)
    print(confusion_matrix(label_number, pred_number))
    print(mean_absolute_error(label_number, pred_number))
    print('--------')
    i+=1

    # tentamos de tudo, mas não está funcionando o gráfico da curva roc
    # plot_roc_curve(classifier, x_test, pred_number)
```

KNN

```
# KNN Treino
from sklearn.neighbors import KNeighborsClassifier
# inicialização do classificador -> configuração para 3 vizinhos.
clf_knn = KNeighborsClassifier(n_neighbors=3)
# treino do classificador
clf_knn.fit(train_features_norm, train_label)
# previsão da classe de teste
test_pred = clf_knn.predict(test_features_norm)
train_pred = clf_knn.predict(train_features_norm)
# print train pred and real labels shape
print(train_pred.shape, train_label.shape)
```

```
(113224,) (113224,)
```

```
# Erro - KNN - Treino
```

```
fn_mean_absolute_error(train_label, train_pred)
```

```
0.0009273652229209355
```

```
# Matriz de Confução - KNN - Treino
fn_cunfusion_matrix(train_label, train_pred)
```

```
[[76488    13     0     0]
 [   10 21912    14     1]
 [    4    53  1333     0]
 [    1     1     0 13394]]
```

```
# Precisão - KNN - Treino
fn_precision_score(train_label,train_pred)
```

```
0.9991432911750159
```

```
# Curva Roc - KNN - Treino
fn_roc_curve(train_label, train_pred)
```

```
(array([0.        , 0.14672407, 0.16132637, 0.16206032, 1.        ]),
 array([0.00000000e+00, 4.55850846e-05, 6.83776268e-04, 9.99544149e-01,
        1.00000000e+00]))
```

```
# Gráfico curva ROC KNN - Treino
# fn_roc_curve_plot()
```

```
# Cross validation correto já plotando o gráfico de roc para cada uma das pastas
fn_KFold(clf_knn, train_features_norm, train_label)
```

```
kfold[ 1 ]:
0.9980128063590197
[[22591     1     0    42]
 [    0     9     0     0]
 [    0     2     0     0]
 [    0     0     0     0]]
0.005696621770810334
--------
kfold[ 2 ]:
0.9994700816957386
[[22622     1    11]
 [    0    11     0]
 [    0     0     0]]
0.0015014351954073747
--------
kfold[ 3 ]:
0.8704349746080813
[[16043     0     0     0]
 [    0   299    11     0]
 [    0     0     0     0]
 [  977  1928    18  3369]]
0.30099359682049015
--------
kfold[ 4 ]:
0.8429233826451755
```

```
[[ 5248     0     0     1]
 [ 2532 12739   156   678]
 [   56    70   372     2]
 [   37    10    15   729]]
0.19328770147935526
--------
kfold[ 5 ]:
0.9376876876876877
[[9898   43    0    0]
 [ 377 4929   12  184]
 [ 295  161  159  273]
 [  11   55    0 6247]]
0.08686627804274863
--------
```

```
# Vizinhos KNN
from sklearn.neighbors import KNeighborsClassifier
# inicialização do classificador -> configuração para 3 vizinhos.
clf_knn = KNeighborsClassifier(n_neighbors=3)
# teste do classificador
clf_knn.fit(test_features_norm, test_label)
# previsão da classe de teste
test_pred_nei = clf_knn.predict(test_features_norm)
# print test pred and real labels shape
print(test_pred_nei.shape, test_label.shape)
```

```
(28306,) (28306,)
```

```
# Erro - KNN - Teste
fn_mean_absolute_error(test_label, test_pred)
```

```
0.3313431781247792
```

```
# Matriz de Confução - KNN - Teste
fn_cunfusion_matrix(test_label, test_pred)
```

```
[[16854     1     0     0]
 [    0  1926     0     0]
 [    0     1     1     0]
 [ 2592   751    99  6081]]
```

```
# Precisão - KNN - Teste
fn_precision_score(test_label,test_pred)
```

```
0.8783296827527732
```

```
# Curva Roc - KNN - Teste
fn_roc_curve(test_label,test_pred)
```

```
(array([0.        , 0.23051554, 0.23430629, 0.26285064, 1.        ]),
 array([0., 0., 0., 1., 1.]))
```

```
#Gráfico curva ROC KNN - Teste
# fn_roc_curve_plot()
```

```
# fn_roc_curve_plot()
```

```
fn_KFold(clf_knn, test_features_norm, test_label)
```

```
    kfold[ 1 ]:
    0.9242317202401978
    [[  68    0    0    0]
     [  39  416    0  327]
     [   1    0    0    1]
     [  61    0    0 4749]]
    0.15524549629106324
    --------
    kfold[ 2 ]:
    0.9395866454689984
    [[ 896    0    0]
     [  20   32    1]
     [ 253   68 4391]]
    0.1619855149266914
    --------
    kfold[ 3 ]:
    1.0
    [[5661]]
    0.0
    --------
    kfold[ 4 ]:
    0.9938173467585232
    [[5625   35]
     [   0    1]]
    0.018547959724430314
    --------
    kfold[ 5 ]:
    0.991697579932874
    [[4523    1   46]
     [   0 1091    0]
     [   0    0    0]]
    0.02455396573043632
    --------
```

Random Forest

```
# Random Forest Treino
from sklearn.ensemble import RandomForestClassifier
# inicializa classificador
clf_rf = RandomForestClassifier(n_estimators=10)
# train classifier
clf_rf.fit(train_features_norm, train_label)
# predict test classes
train_pred = clf_rf.predict(train_features_norm)
# print train pred and real labels shape
print(train_pred.shape, train_label.shape)
```

```
    (113224,) (113224,)
```

```
# Erro - Random Forest - Treino
fn_mean_absolute_error(train_label, train_pred)
```

```
0.0001501448456157705
```

```
# Matriz de Confução - Random Forest - Treino
fn_cunfusion_matrix(train_label, train_pred)
```

```
[[76499     2     0     0]
 [    1 21936     0     0]
 [    0    14  1376     0]
 [    0     0     0 13396]]
```

```
# Precisão - Random Forest - Treino
fn_precision_score(train_label, train_pred)
```

```
0.9998498551543842
```

```
# Curva Roc - Random Forest  - Treino
fn_roc_curve(train_label, train_pred)
```

```
(array([0.        , 0.14674598, 0.16181932, 0.16199459, 1.        ]),
 array([0.        , 0.        , 0.        , 0.99995441, 1.        ]))
```

```
# Gráfico curva  ROC - Random Forest - Treino
# fn_roc_curve_plot()
```

```
# Cross validation correto já plotando o gráfico de roc para cada uma das pastas
fn_KFold(clf_rf, train_features_norm, train_label)
```

```
kfold[ 1 ]:
0.9999116802826231
[[22634     0     0]
 [    0     9     0]
 [    0     2     0]]
8.83197173769044e-05
--------
kfold[ 2 ]:
1.0
[[22634     0]
 [    0    11]]
0.0
--------
kfold[ 3 ]:
0.9013910355486863
[[16043     0     0     0]
 [    0   298    12     0]
 [    0     0     0     0]
 [    4  2153    64  4071]]
0.19403841907705896
--------
kfold[ 4 ]:
0.8776771914329874
[[ 5249     0     0     0]
 [   12 13514    17  2562]
 [    0    79   334    87]
 [    3     7     3   778]]
```

```
0.236034444689777
--------
kfold[ 5 ]:
0.9248807631160573
[[9862   78    0    1]
 [   2 4725   19  756]
 [  39   30   99  720]
 [   0   56    0 6257]]
0.11278925984808338
--------
```

```
# Random Forest - Teste
from sklearn.ensemble import RandomForestClassifier
# inicializa classificador
clf = RandomForestClassifier(n_estimators=10)
# train classifier
clf.fit(test_features_norm, test_label)
# predict test classes
test_pred = clf.predict(test_features_norm)
# print test pred and real labels shape
print(test_pred.shape, test_label.shape)
```

```
(28306,) (28306,)
```

```
#Erro - Random Forest - Teste
fn_mean_absolute_error(test_label, test_pred)
```

```
0.0
```

```
# Matriz de Confução - Random Forest - Teste
fn_cunfusion_matrix(test_label,test_pred)
```

```
[[16855     0     0     0]
 [    0  1926     0     0]
 [    0     0     2     0]
 [    0     0     0  9523]]
```

```
# Precisão - Random Forest - Teste
fn_precision_score(test_label,test_pred)
```

```
1.0
```

```
# Curva Roc - Random Forest - Teste
fn_roc_curve(test_label,test_pred)
```

```
(array([0.        , 0.36099318, 0.36106899, 0.36106899, 1.        ]),
 array([0., 0., 0., 1., 1.]))
```

```
# Gráfico curva ROC -Random Forest - Teste
# fn_roc_curve_plot()
```

```
fn_KFold(clf_rf, test_features_norm, test_label)
```

```
kfold[ 1 ]:
0.9851642529141647
[[   0    0    0   68]
 [   0  768    0   14]
 [   0    2    0    0]
 [   0    0    0 4810]]
0.04132815259625574
--------
kfold[ 2 ]:
0.9991167638226461
[[ 896    0    0]
 [   1   48    4]
 [   0    0 4712]]
0.001589825119236884
--------
kfold[ 3 ]:
1.0
[[5661]]
0.0
--------
kfold[ 4 ]:
1.0
[[5660    0]
 [   0    1]]
0.0
--------
kfold[ 5 ]:
1.0
[[4570    0]
 [   0 1091]]
0.0
--------
```

MLP

```
#MLP Treino
from sklearn.neural_network import MLPClassifier
clf_mlp = MLPClassifier(solver='lbfgs', max_iter=200)
clf_mlp.fit(train_features_norm, train_label)
train_pred = clf_mlp.predict(train_features_norm)
print(train_pred.shape, train_label.shape)
```

```
(113224,) (113224,)
```

```
# Matriz de Confusão - MLP - Treino
fn_cunfusion_matrix(train_label, train_pred)#(train_label, train_pred)
```

```
[[76477    24     0     0]
 [   11 21922     4     0]
 [    3    79  1308     0]
 [    0     0     0 13396]]
```

```
# Precisão - MLP - Treino
```

```
fn_precision_score(train_label, train_pred)
```

```
0.9989313219812054
```

```
#Erro - MLP - Teste
fn_mean_absolute_error(train_label, train_pred)
```

```
0.0010951741680209142
```

```
# Curva Roc - MLP  - Treino
fn_roc_curve(train_label, train_pred)
```

```
(array([0.        , 0.14674598, 0.16107441, 0.16220272, 1.        ]),
 array([0.00000000e+00, 0.00000000e+00, 1.82340338e-04, 9.99498564e-01,
        1.00000000e+00]))
```

```
# Gráfico curva  ROC - MLP - Treino
# fn_roc_curve_plot()
```

```
# Cross validation correto já plotando o gráfico de roc para cada uma das pastas
fn_KFold(clf_mlp, train_features_norm, train_label)
```

```
kfold[ 1 ]:
0.9994700816957386
[[22624     0     0    10]
 [    0     9     0     0]
 [    0     2     0     0]
 [    0     0     0     0]]
0.0014131154780304704
--------
kfold[ 2 ]:
0.9999558401413116
[[22633     0     1]
 [    0    11     0]
 [    0     0     0]]
0.0001324795760653566
--------
kfold[ 3 ]:
0.8092294104658865
[[16043     0     0     0]
 [    0   310     0     0]
 [    0     0     0     0]
 [   38  3997   285  1972]]
0.37063369397217927
--------
kfold[ 4 ]:
0.9337160521086333
[[ 5248     0     0     1]
 [    3 14763  1325    14]
 [    0    87   378    35]
 [    0    22    14   755]]
0.06796202252152793
--------
kfold[ 5 ]:
0.9367602897014662
[[9897   44    0    0]
```

```
 [   3 4947   60  492]
 [  11  215   85  577]
 [   1    0   29 6283]]
0.08554142377671789
--------
```

```
#MLP Teste
from sklearn.neural_network import MLPClassifier
clf_mlp = MLPClassifier(solver='lbfgs', max_iter=200)
clf_mlp.fit(test_features_norm, test_label)
test_pred = clf_mlp.predict(test_features_norm)
print(test_pred.shape, test_label.shape)
```

```
(28306,) (28306,)
```

```
#Erro - MLP - Teste
fn_mean_absolute_error(test_label, test_pred)
```

```
0.00017664099484208294
```

```
# Matriz de Confusão - MLP - Teste
fn_cunfusion_matrix(test_label,test_pred)
```

```
[[16855     0     0     0]
 [    1  1924     0     1]
 [    0     2     0     0]
 [    0     0     0  9523]]
```

```
# Precisão - MLP - Teste
fn_precision_score(test_label,test_pred)
```

```
0.9998586872041263
```

```
# Curva Roc - MLP - Teste
fn_roc_curve(test_label,test_pred)
```

```
(array([0.        , 0.36099318, 0.36106899, 1.        ]),
 array([0.00000000e+00, 5.19210800e-04, 9.99480789e-01, 1.00000000e+00]))
```

```
# Gráfico curva ROC - MLP - Teste
# fn_roc_curve_plot()
```

```
fn_KFold(clf_mlp, test_features_norm, test_label)
```

```
kfold[ 1 ]:
0.9418933239138114
[[  68    0    0    0]
 [   0  456    0  326]
 [   0    1    0    1]
 [   1    0    0 4809]]
0.11603673613564111
--------
kfold[ 2 ]:
```

```
0.9977035859388801
[[ 896    0    0]
 [   1   52    0]
 [   0   12 4700]]
0.004416180886769122
--------
kfold[ 3 ]:
1.0
[[5661]]
0.0
--------
kfold[ 4 ]:
1.0
[[5660    0]
 [   0    1]]
0.0
--------
kfold[ 5 ]:
0.9994700582935877
[[4567    0    3]
 [   0 1091    0]
 [   0    0    0]]
0.001589825119236884
--------
```

Avaliação de Resultados

Para criação dos modelos, foi utilizada uma base de treinamento contendo 80% dos registros do dataset, corespondendo a 113.224 registros. A avaliação dos modelos ocorreu com a execução dos 20% restantes do dataset, ou seja 28.306 registros

Os seguintes resultados foram obtidos com a execução dos algoritmos KNN, Ramdom Forest e a Rede Neural MLP

.	Erro	Matriz de Confusão	Precisão	Curva Roc
KNN Treino	0.09%	[76488.....13.....0.....0] [....10....21912.....14.....1] [.....4.....53.....1333..... 0] [.....1.....1.....0..13394]	99.91%	(array([0, 0.14672407, 0.16132637, 0.16206032, 1]), array([0.00000000e+00, 4.55850846e-05, 6.83776268e-04, 9.99544149e-01, ...1.00000000e+00]), array([5., 4., 3., 2., 1.]))
KNN Teste	33.13%	[16854.....1..... 0..... 0] [.....0.....1926..... 0..... 0] [.....0.....1.....1..... 0] [2592.....751.....99....6081]	87.83%	(array([0, 0.23051554, 0.23430629, 0.26285064, 1]), array([0., 0., 0., 1., 1.]), array([5., 4., 3., 2., 1.]))

.	Erro	Matriz de Confusão	Precisão	Curva Roc
Ramdom Forest Treino	0.02%	[76497.....4..... 0..... 0] [.....0.....21937..... 0..... 0] [.....2.....18.....1370.....0] [.....0..... 0..... 0...13396]	99.98%	(array([0, 0.14674598, 0.16170977, 0.16197268, 1]), array([0, 0, 0, 0.99995441, 1]), array([5., 4., 3., 2., 1.]))
Random Forest Teste	0.01%	[16855.....0.....0..... 0] [.....0.....1926.... 0..... 0] [.....0.....0..... 2..... 0] [.....0.....0..... 0...9523]	99.99%	(array([0, 0.36099318, 0.36103108, 0.36106899, 1]), array([0., 0., 0., 1., 1.]), array([5., 4., 3., 2., 1.]))
MLP Treino	0.109%	[76446.....31.....11.....13] [.....0.....21930..... 0..... 7] [....11.....739.....623.....17] [.....1.....9651.....51.....3693]]	99.89%	(array([0, 0.14674598, 0.16107441, 0.16220272, 1]), array([0.00000000e+00, 0.00000000e+00, 1.82340338e-04, 9.99498564e-01, ... 1.00000000e+00]), array([5., 4., 3., 2., 1.]))
MLP Teste	0.018%	[16855.....0..... 0..... 0] [.....0....1926..... 0..... 0] [.....0..... 1..... 1..... 0] [.....0..... 0..... 0....9523]]	99.98%	(array([0, 0.36099318, 0.36106899, 0.36106899, 1]), array([0., 0., 5.19210800e-04, 9.99480789e-01, 1.]), array([5., 4., 3., 2., 1.]))

Na análise dos resultados dos parâmetros executados para cada algoritmo, percebeu-se que o Ramdon Forest apresentou melhor precisão, menor erro e mais adequada classificação dos registros. O KNN apresentou resultado satisfatório, próximo ao do Random Forest durante o treinamento, mas o resultado do teste foi inferior. Além disso foi percebido também que o KNN apresentou bastante lentidão quando executado para uma grande base de dados. O MLP pelo fato de ser sensível à escalabilidade, não se mostrou muito efeciente no dataset de entrada. Uma possível explicação para isso, seria que temos uma grande quantidade de atributos que levou a muitos erros na classificação. O projeto desenvolvido foi bastante desafiador, sendo que os resultados obtidos foram considerados satisfatórios.