Computação aplicada - Inteligência computacional

Detecção de perdas não técnicas em sistemas de distribuição de energia e análise de desempenho entre modelos de aprendizagem de máquina utilizando dados de vizinhança

Universidade Federal do Pará (UFPA).

Instituto de Tecnologia (ITEC).

Programa de Pós-graduação em engenharia elétrica (PPGEE).

Orientador: Marcelino Silva da Silva.

Coorientador: Roberto Célio Limão de Oliveira.

Discente: Thiago Furtado Ferreira.

Objetivo da modelagem

Determinar quais consumidores de energia elétrica possuem defeito na medição sejam por natureza técnica do equipamento ou por fraude proposital e consumidores que estão com medição normal.

A base de dados foi processada com normalização MIN-MAX, 200 variaveis de entrada, 1 de targer e outra de ID, totalizando 202 variaveis.

O target possui 2 classes, 0 e 1, sendo:

- 0 - Consumidores que foram fiscalizados e que não possuiam fraude ou defeito na medição.
- 1 - Consumidores fiscalizados que possuiam fraude ou defeito na medição.

A base de dados foi gerada com restrições com finalidade de limpar os dados.

- Foram considerados somente os consumidores de baixa tensão (BT).
- Consumidores BT com ligação em média tensão foram removidos, pois seu consumo é comparavel com um consumidor AT (Alta tensão).
- Pontos de medição fora dos limites da área de concessão da concessionária de energia foram removidos, informações baseadas nas coordenadas geográficas dos pontos de medição.
- Consumidores BT com consumo mensal maior ou igual a 20000 KWH foram removidos.
- Registros com valores nulos foram removidos.
- Registros com consumo negativo foram removidos.
- Removidas todos os registros com data de ligação superior ao dia atual de apuração.

A partir dos dados de coordenadas dos consumidores foi feito um processamento para verificar quais outros consumidores estavam dentro de um determinado raio para geração de outras variáveis dos vizinhos que a base de dados original não contempla.

- O raio considerado foi 30 metros do ponto de medição do consumidor.
- Número de fraudes, número de fiscalizações, total de inspeções normais, consumo máximo, consumo minino, quantidade total de consumos minimos da fase em 24 meses, fraudes exclusivas por cliente, quantidade de pontos de medição no raio.
- Consumidores com mais de 250 pontos de medição vizinho no raio de 30 metros foram descartados.

Da divisão das bases de dados

Os dados foram divididos em 5 bases, sendo:

- Uma base de dados para treinamento balanceada, com 50 mil registros, 25 mil normais e 25 mil fraude.
- Uma base de dados para score do modelo balanceada com 43396 registros no total.
- Uma base balanceada para validação cruzada com 11572 registros.
- Uma base balanceada para teste do modelo com 17358 registros.
- Uma base para teste desbalanceada positiva com 11163 registros, sendo 92% dos registros de fraude e 8% de normais.
- Uma base para teste desbalanceada negativa com 11163 registros, sendo 92% dos registros normais e 8% com fraude.

Nenhum dos registros se repete em nenhuma das bases.

Importação das bibliotecas

In [1]:

```python

#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
import pandas as pd
import numpy as np
from datetime import datetime
import string
import os
from sklearn.model_selection import cross_val_score as validacaoCruzada
from sklearn.metrics import confusion_matrix as matrizConfusao
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.tree import export_graphviz
from sklearn import preprocessing
import seaborn as sn
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.cluster import KMeans
from sklearn.svm import SVC
from sklearn import metrics
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import validation_curve
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
### Configurações para baixar dados do Google Drive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
#from google.colab import auth
#from google.colab import files
#from oauth2client.client import GoogleCredentials
#auth.authenticate_user()
#gauth = GoogleAuth()
#gauth.credentials = GoogleCredentials.get_application_default()
#drive = GoogleDrive(gauth)
import logging
logging.getLogger('googleapiclient.discovery_cache').setLevel(logging.ERROR)
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
### Variaveis globais
resultadoValidacaoCruzada=""
vp=0
vn=0
fp=0
fn=0
vetorResultadoTeste=[];
vetorResultadoCV=[];
vetorScore=[];
vetorMSETeste=[];
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
### Função para gravar log e exibir mensagens em tela
def m(conteudo, exibir):
    hora = datetime.now()
    hora = hora.strftime("%d/%m/%Y_%H:%M:%S")

#    from google.colab import files
#    with open(caminhoArquivoLog, 'a') as f:
    conteudo_ = "[" + hora + "]: "+ conteudo
#      f.write(conteudo_  + " \n")
#    f.close()
#    if exibir == 1:
    print(conteudo_)

#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
### Arquivos de dados
#LISTA OS ARQUIVOS DA PASTA DO GOOGLE DRIVE
#file_list = drive.ListFile({'q': "'1EintRF9-o9c0nKiNVKedXXSq7-I4Vvj2' in parents and trashed=false"}).GetList()
##for file1 in file_list:
#  print('title: %s, id: %s' % (file1['title'], file1['id']))
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
### Gera Histograma
def geraHistograma(dtHistograma, caption):
  dtHistograma_=dtHistograma['TARGET']
  dtHistograma_=dtHistograma_.to_numpy()
  histograma=pd.DataFrame(dtHistograma_, columns = [caption])
  histograma.hist(bins=3)
 # pc()
```

In [2]:
```
d=4
print(d)
```

```
4
```

In [3]:
```
dtTreino = pd.read_excel(r'C:\Users\Computador\Documents\dadosExecucaoLocal\A50000k_TREINO_NORMALIZADA.xlsx', sheet_r
#geraHistograma(dtTreino, 'Histograma do Target da base de Treino')
dtScore = pd.read_excel(r'C:\Users\Computador\Documents\dadosExecucaoLocal\B43396k_SCORE_NORMALIZADA.xlsx', sheet_nam
#geraHistograma(dtScore, 'Histograma do Target da base de Score')
dtValidacao = pd.read_excel(r'C:\Users\Computador\Documents\dadosExecucaoLocal\C11572K_VALIDACAO_NORMALIZADA.xlsx', s
#geraHistograma(dtValidacao, 'Histograma do Target da base de Validação')
dtTeste = pd.read_excel(r'C:\Users\Computador\Documents\dadosExecucaoLocal\C17358K_TESTE_NORMALIZADA.xlsx', sheet_nam
#geraHistograma(dtTeste, 'Histograma do Target da base de Teste')
dtTestePositivo = pd.read_excel(r'C:\Users\Computador\Documents\dadosExecucaoLocal\A11163_DESBALANCO_POSITIVO_NORMALI
#geraHistograma(dtTestePositivo, 'Histograma do Target da base de Teste Positivo. 92% positivo e 8% negativo')
dtTesteNegativo = pd.read_excel(r'C:\Users\Computador\Documents\dadosExecucaoLocal\A11163_DESBALANCO_NEGATIVO_NORMALI
#geraHistograma(dtTesteNegativo, 'Histograma do Target da base de Teste Negativo. 92% negativo e 8% positivo.')

```

Carrega os dados

In [5]:

```
### Função para computar os resultados da validação cruzada
def exibeResultados(scores, exibir=1):
    if (exibir==1):
      contador=0
      for score in scores:
          print("Score "+ str(contador) +": " + str(score))
          contador=contador+1
      print("Máximo: " + str(scores.max()))
      print("Minimo: " + str(scores.min()))
      print("Média: " + str(scores.mean()))
      print("Desv. Padrão: " + str(scores.std()))
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
### ETL - Ajuste de dados
#Separa os dados de target dos demais dados de entrada
yTreino = dtTreino['TARGET']
XTreino = dtTreino.drop(['INSTALACAO','TARGET', 'REINCIDENCIA'],axis=1)

yScore = dtScore['TARGET']
XScore = dtScore.drop(['INSTALACAO','TARGET', 'REINCIDENCIA'],axis=1)

yValidacao = dtValidacao['TARGET']
XValidacao = dtValidacao.drop(['INSTALACAO','TARGET', 'REINCIDENCIA'],axis=1)

yTeste = dtTeste['TARGET']
XTeste = dtTeste.drop(['INSTALACAO','TARGET', 'REINCIDENCIA'],axis=1)

yTestePositivo = dtTestePositivo['TARGET']
XTestePositivo = dtTestePositivo.drop(['INSTALACAO','TARGET', 'REINCIDENCIA'],axis=1)

yTesteNegativo = dtTesteNegativo['TARGET']
XTesteNegativo = dtTesteNegativo.drop(['INSTALACAO','TARGET', 'REINCIDENCIA'],axis=1)
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
### Gera gráficos, Matriz de confusão e ROC-UAC
def geraGraficos(vp_, vn_, fp_, fn_, yReal__, yModelo__, titulo__):
  #GERA MATRIZ DE CONFUSAO
  array = [[vp_,fp_],[fn_,vn_]]
  df_cm = pd.DataFrame(array,index =['FRAUDE', 'NORMAL'],  columns = ['FRAUDE', 'NORMAL'])
  ax=sn.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt="d", cbar=False)
  ax.set_title('Matriz de confusão ' + titulo__, weight="bold")
```

```python
    plt.xlabel('Real', weight="bold", size=18)
    plt.ylabel('Modelo',weight="bold", size=18)
    plt.figure(figsize = (5,3))
    plt.show()


    #GERA ROC-UAC
    yReal__ = np.array(yReal__)
    yModelo__ = np.array(yModelo__)
    roc_auc_score(yReal__, yModelo__)
    fpr, tpr, _ = metrics.roc_curve(yReal__,  yModelo__)
    auc = metrics.roc_auc_score(yReal__, yModelo__)
    ax2=plt.plot(fpr,tpr,label="AUC = "+str(auc))
    plt.title('Curva ROC ' + titulo__)
    plt.legend(loc=4)
    plt.show()

    return auc
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
## Converte para Array
#Converte os dataFrames com os dados carregados para array NumPy.

XTreino=XTreino.to_numpy()
yTreino=yTreino.to_numpy()

XScore=XScore.to_numpy()
yScore=yScore.to_numpy()

XValidacao=XValidacao.to_numpy()
yValidacao=yValidacao.to_numpy()

XTeste=XTeste.to_numpy()
yTeste=yTeste.to_numpy()

XTestePositivo=XTestePositivo.to_numpy()
yTestePositivo=yTestePositivo.to_numpy()

XTesteNegativo=XTesteNegativo.to_numpy()
yTesteNegativo=yTesteNegativo.to_numpy()
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```python
### Adiciona resultados dos teste ao vetor final para exportar
def addExportar(metrica__, valor__, algoritmo__, tipoTeste__):
  global vetorResultadoTeste
  resultadosProcessados__=[metrica__, valor__, algoritmo__, tipoTeste__];
  vetorResultadoTeste.append(resultadosProcessados__)
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
## Adiciona os Score para o BloxPlot
def addBoxPLot(algoritmo___,score___):
  global vetorResultadoCV
  resultadosProcessados___=[algoritmo___,score___];
  vetorResultadoCV.append(resultadosProcessados___)
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
### Computa as métricas dos testes
#Além das métricas dos teste a função gera um arquivo txt com os dados.
def computaMetricas(apuracao, nomeAlgoritmo, nomeTeste):
  acertos=apuracao['ACERTOS'].sum()
  erros=apuracao['ERROS'].sum()
  global vp
  vp=apuracao['VP'].sum()
  global vn
  vn=apuracao['VN'].sum()
  global fp
  fp=apuracao['FP'].sum()
  global fn
  fn=apuracao['FN'].sum()

  sensibilidade=vp/(vp+fn)
  especificidade=vn/(fp+vn)
  precisao=vp/(vp+fp)
  acuracia=(vp+vn)/(vp+vn+fp+fn)
  revocacao=vp/(vp+fn)
  pontuacao_f1=vp/(vp+((fn+fp)/2))

  addExportar('SENSIBILIDADE', float(sensibilidade), nomeAlgoritmo, nomeTeste)
  addExportar('ESPECIFICIDADE', float(especificidade), nomeAlgoritmo, nomeTeste)
  addExportar('PRECISAO', precisao, nomeAlgoritmo, nomeTeste)
  addExportar('ACURACIA', acuracia, nomeAlgoritmo, nomeTeste)
  addExportar('REVOCACAO', revocacao, nomeAlgoritmo, nomeTeste)
  addExportar('PONTUACAO_F1', pontuacao_f1, nomeAlgoritmo, nomeTeste)
  addExportar('VP', vp, nomeAlgoritmo, nomeTeste)
```

```
  addExportar('VN', vn, nomeAlgoritmo, nomeTeste)
  addExportar('FP', fp, nomeAlgoritmo, nomeTeste)
  addExportar('FN', fn, nomeAlgoritmo, nomeTeste)
  addExportar('TOTAL REGISTROS', vp+vn+fp+fn, nomeAlgoritmo, nomeTeste)
  addExportar('ACERTOS', acertos, nomeAlgoritmo, nomeTeste)
  addExportar('ERROS', erros, nomeAlgoritmo, nomeTeste)
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

Funções com modelo

In [6]:

```
### Executa Treino
def executaTreino(mdl_treino):
 # pi()
  mdl_treino.fit(XTreino, yTreino)

### Executa Score
def executaScore(mdl_score, nomeAlgoritmo):
  score=mdl_score.score(XScore, yScore)
  global vetorScore
  vetorScore__=[nomeAlgoritmo, score];
  vetorScore.append(vetorScore__)
  return score

### Executa Validação Cruzada
def executaValidacaoCruzada(mdl_validacaoCruzada, folds, metrica, nomeAlgoritmo___):
  scores = validacaoCruzada(mdl_validacaoCruzada, XValidacao, yValidacao, cv=folds,scoring=metrica)
  addBoxPLot(nomeAlgoritmo___, scores)
  return scores.mean()

### Executa Teste
def executaTeste(mdl_testar, xTeste__, yTeste__, nomeAlgoritmo, tipoTeste):
  resultadoTeste=mdl_testar.predict(xTeste__)
  dtResultadoTeste=pd.DataFrame(resultadoTeste)
  dtTarget=pd.DataFrame(yTeste__)

  dtExportar=pd.concat([dtTarget, dtResultadoTeste, (dtTarget==dtResultadoTeste)*1, (dtTarget!=dtResultadoTeste)*1, (
  dtExportar.columns = ['TARGET', 'MODELO', 'ACERTOS','ERROS', 'VP', 'VN', 'FP','FN']
  computaMetricas(dtExportar, nomeAlgoritmo, tipoTeste)
  titulo= nomeAlgoritmo + ' ' + tipoTeste
  uac=geraGraficos(vp,vn,fp, fn, yTeste__, resultadoTeste, titulo)
  return uac

## Função Principal
def funcMain(modelo, nome, folds___, metrica___):
  from sklearn import preprocessing
  XTesteAA=XTeste
  XTestePositivoAA=XTestePositivo
  XTesteNegativoAA=XTesteNegativo

  executaTreino(modelo)
  rs_vc=executaValidacaoCruzada(modelo, folds___, metrica___, nome)
```

```
42
43   executaScore(modelo,nome)
44   uac1=executaTeste(modelo, XTesteAA, yTeste, nome, 'TESTE')
45   uac2=executaTeste(modelo, XTestePositivoAA, yTestePositivo, nome + '+', 'POSITIVO')
46   uac3=executaTeste(modelo, XTesteNegativoAA, yTesteNegativo, nome + '-', 'NEGATIVO')
47   print(rs_vc)
48   print(uac1)
49   print(uac2)
50   print(uac3)
51
```

Parametros

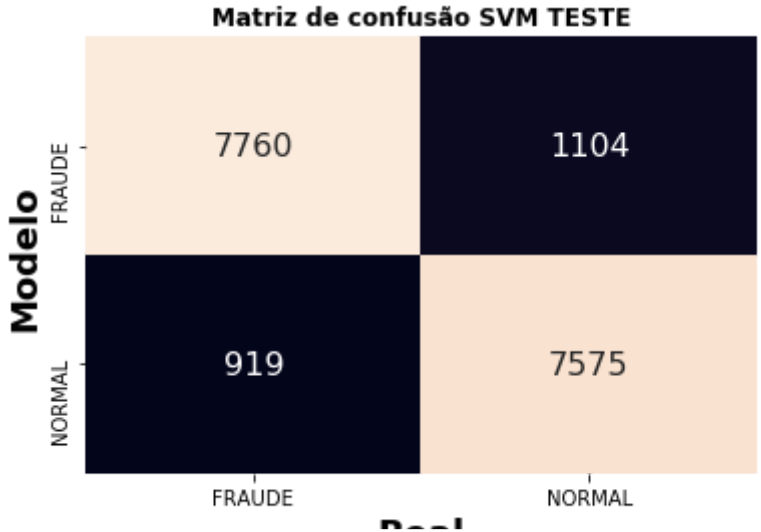
In [1]:
```
1  nfoldsCV=5
2  tpMetricaCV='precision'
```

1-SVM

In [13]:
```
mdl_svm = SVC(C=1.0, kernel='poly', degree=5, gamma='scale', coef0=0.5, shrinking=True,
              probability=False, tol=1e-03, cache_size=5000, class_weight=None, verbose=False,
              max_iter=25000, decision_function_shape='ovr',random_state=None)

funcMain(mdl_svm,'SVM', nfoldsCV, tpMetricaCV)
```

```
c:\users\computador\anaconda3\lib\site-packages\sklearn\svm\base.py:244: ConvergenceWarning: Solver terminated early
(max_iter=25000).  Consider pre-processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
```



2-KMEANS

2.1-Treinamento Kmeans

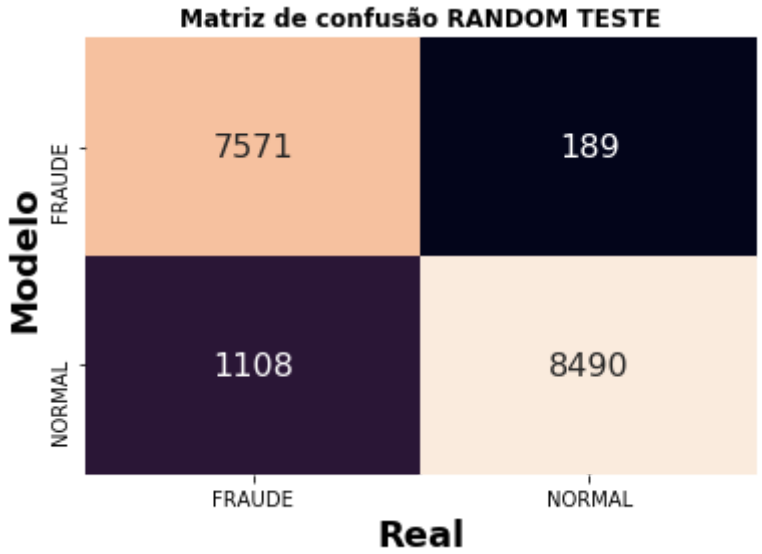
In [19]:
```
mdl_kmeans = KMeans(n_clusters=2, init='k-means++', n_init=10, max_iter=300, tol=0.0001,
                    precompute_distances='auto', verbose=0, random_state=None,
                    copy_x=True, n_jobs=None, algorithm='auto')


#funcMain(mdl_kmeans,'KMEANS', nfoldsCV, tpMetricaCV)

```

3-Random Forest

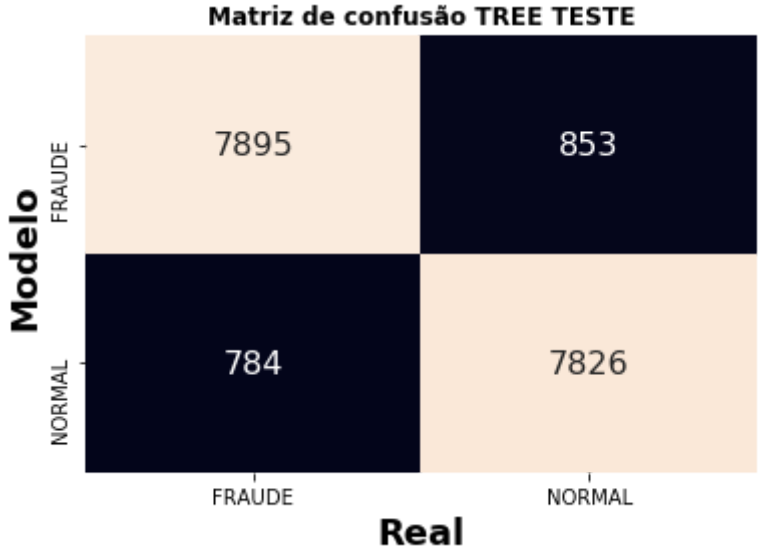
In [20]:
```
mdl_rndForest = RandomForestClassifier(n_estimators=500, criterion='entropy', max_depth=50, min_samples_split=2,
                                       min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='sqrt',
                                       max_leaf_nodes=None, min_impurity_decrease=1e-09,
                                       bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0,
                                       warm_start=False, class_weight=None)
funcMain(mdl_rndForest,'RANDOM',nfoldsCV, tpMetricaCV)
```




```
<Figure size 360x216 with 0 Axes>
```

4-Decision Tree

In [22]:
```
1 mdl_decisionTree = tree.DecisionTreeClassifier(criterion='entropy', max_depth=300, splitter='best', min_impurity_decr
2 funcMain(mdl_decisionTree,'TREE', nfoldsCV, tpMetricaCV)
```




```
<Figure size 360x216 with 0 Axes>
```

5-MLP

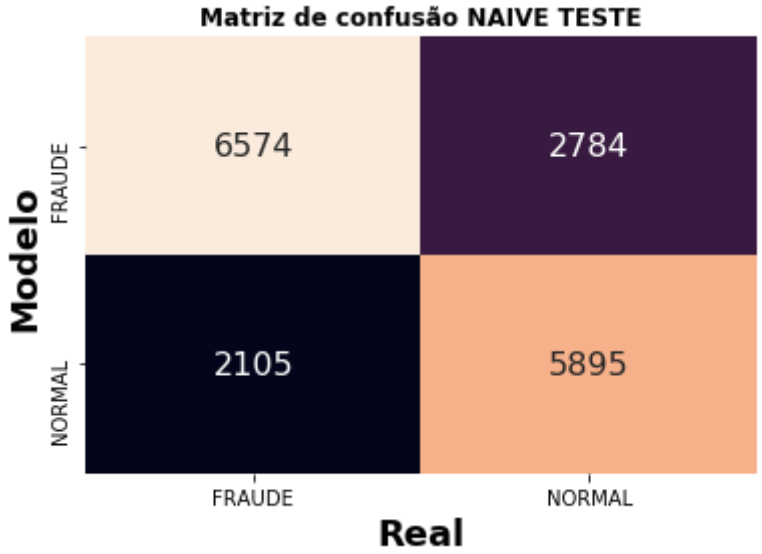
In [24]:

```
mdl_mlp = MLPClassifier(hidden_layer_sizes=(93), activation='relu',
                        solver='adam', alpha=1e-04, batch_size='auto', learning_rate='constant',
                        learning_rate_init=0.001, power_t=0.5, max_iter=350, shuffle=True, random_state=None, tol=1e-
                        verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False,
                        validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10)
funcMain(mdl_mlp,'MLP', nfoldsCV, tpMetricaCV)
```

```
c:\users\computador\anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:562: ConvergenceWarni
ng: Stochastic Optimizer: Maximum iterations (350) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
c:\users\computador\anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:562: ConvergenceWarni
ng: Stochastic Optimizer: Maximum iterations (350) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
c:\users\computador\anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:562: ConvergenceWarni
ng: Stochastic Optimizer: Maximum iterations (350) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
c:\users\computador\anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:562: ConvergenceWarni
ng: Stochastic Optimizer: Maximum iterations (350) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

6-Naive

In [8]:
```
1
2 mdl_naive=clf = GaussianNB(priors=None, var_smoothing=1e-17)
3 funcMain(mdl_naive,'NAIVE', nfoldsCV, tpMetricaCV)
```

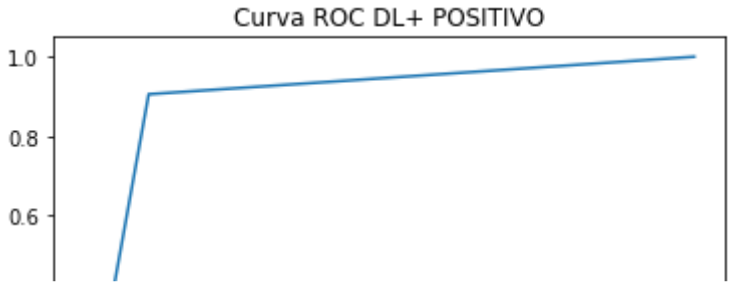



```
<Figure size 360x216 with 0 Axes>
```

7-Deep Learning

In [9]:
```
mdl_dl = MLPClassifier(hidden_layer_sizes=(400,400,400,400,400,400,400,400,400),
                       activation='relu',
                        solver='adam', alpha=1e-04, batch_size='auto', learning_rate='constant',
                        learning_rate_init=1e-03, power_t=0.5, max_iter=350, shuffle=True, random_state=None, tol=1e-
                        verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False,
                        validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10)
funcMain(mdl_dl,'DL', nfoldsCV, tpMetricaCV)
```


```
<Figure size 360x216 with 0 Axes>
```



Exporta os dados dos testes já realizados

In [27]:
```
dtModelo=pd.DataFrame(vetorResultadoTeste)

dtModelo.columns = ['METRICA', 'VALOR', 'MODELO','TIPO_TESTE']
dataAtual= datetime.now()
dataAtual = dataAtual.strftime("_%d_%m_%Y_%H_%M_%S")
nomeArquivo___='E:\\resultados\\metricas'+ dataAtual +'.xlsx'
#from google.colab import files
dtModelo.to_excel(nomeArquivo___)
#files.download(nomeArquivo___)

```

Gera BoxPlot da validação cruzada

In [ ]:
```
DT__=pd.DataFrame(vetorResultadoCV)
DT__.columns = ['MODELO','VALOR']
fig = plt.figure(figsize=(10, 10))
fig.suptitle('Comparação dos algoritmos CrossValidation')
ax = fig.add_subplot(111)
plt.boxplot(DT__['VALOR'])
ax.set_xticklabels(DT__['MODELO'])
plt.show()
```