Trabalho4

December 7, 2021

0.1 Comando para deixar iopub.data_rate maior que o padrão:

0.1.1 1 - Abra um jupyter notebook com o comando abaixo:

0.1.2 jupyter notebook –NotebookApp.iopub_data_rate_limit=1.0e10

0.2 Pré-processamento

0.2.1 Contando as sequências. Vou usar o software seqkit

[ ]:
```
#lncRNA
!grep ">" basesHumano/gencode.v38.lncRNA_transcripts.fasta | wc -l
```

[ ]:
```
#RNA Transcritos
!grep ">" basesHumano/gencode.v38.pc_transcripts.fasta | wc -l
```

0.2.2 Removendo os ruídos e dados duplicados

[ ]:
```
#lncRNA
!seqkit rmdup -s < basesHumano/gencode.v38.lncRNA_transcripts.fasta >␣
↪basesHumano/lncrna_noduplicado.fasta
```

[ ]:
```
#RNA Transcritos
!seqkit rmdup -s < basesHumano/gencode.v38.pc_transcripts.fasta > basesHumano/
↪rna_trancr_noduplicado.fasta
```

0.3 Usando as funções de pré-processamento do MathFeature

0.3.1 Eliminando ruídos como outras anotações(letras): k,N...

[ ]:
```
#lncRNA
!python3 MathFeature/preprocessing/preprocessing.py -i basesHumano/
↪lncrna_noduplicado.fasta -o basesHumano/lncrna_pre.fasta
```

[ ]:
```
#RNA Transcritos
!python3 MathFeature/preprocessing/preprocessing.py -i basesHumano/
↪rna_trancr_noduplicado.fasta -o basesHumano/rna_pre.fasta
```

0.3.2 Recontanto as sequências

[ ]:
```
#lncRNA
!grep ">" basesHumano/lncrna_pre.fasta | wc -l
```

[ ]:
```
#mRNA Transcritos
!grep ">" basesHumano/rna_pre.fasta | wc -l
```

0.3.3 Executando o sampling para deixa tudo igual

[ ]:
```
#lncRNA
%run MathFeature/preprocessing/sampling.py -i basesHumano/rna_pre.fasta -o␣
↪basesHumano/rna_presampling.fasta -p 97302
```

0.3.4 Recontanto as sequências

[ ]:
```
#lncRNA
!grep ">" basesHumano/lncrna_pre.fasta | wc -l
```

[ ]:
```
#mRNA Transcritos
!grep ">" basesHumano/rna_presampling.fasta | wc -l
```

0.4 Extração de características

0.4.1 OPEN READING FRAME (ORF) DESCRIPTOR

[ ]:
```
#lncRNA
%run MathFeature/methods/CodingClass.py -i basesHumano/lncrna_pre.fasta -o␣
↪basesHumano/ORF_lncrna.csv -l lncRNA
```

[ ]:
```
#mRNA
%run MathFeature/methods/CodingClass.py -i basesHumano/rna_presampling.fasta -o␣
↪basesHumano/ORF_mrna.csv -l mRNA
```

0.4.2 Fickett score

[ ]:
```
#lncRNA
%run MathFeature/methods/FickettScore.py -i basesHumano/lncrna_pre.fasta -o␣
↪basesHumano/FICKETT_lncrna.csv -l lncRNA -seq 1
```

[ ]:
```
#mRNA
%run MathFeature/methods/FickettScore.py -i basesHumano/rna_presampling.fasta -o␣
↪basesHumano/FICKETT_mrna.csv -l mRNA -seq 1
```

0.4.3 Numerical Mapping and Fourier Transform

[ ]:
```
#lncRNA
%run MathFeature/methods/FourierClass.py -i basesHumano/lncrna_pre.fasta -o␣
↪basesHumano/FOURIER_lncrna.csv -l lncRNA -r 2
```

[ ]:
```
#mRNA
%run MathFeature/methods/FourierClass.py -i basesHumano/rna_presampling.fasta -o␣
↪basesHumano/FOURIER_mrna.csv -l mRNA -r 2
```

0.4.4 Complex Networks - desabilitei, pois está demorando mais de um dia para processar.

[ ]:
```
#lncRNA
#%run MathFeature/methods/ComplexNetworksClass.py -i basesHumano/lncrna_pre.
↪fasta -o basesHumano/CN_lncrna.csv -l lncRNA -k 3 -t 5
```

[ ]:
```
#mRNA
#%run MathFeature/methods/ComplexNetworksClass.py -i basesHumano/rna_presampling.
↪fasta -o basesHumano/CN_mrna.csv -l mRNA -k 3 -t 5
```

0.4.5 Extração de características com o iFeature

[ ]:
```
!python iFeature/iFeature.py --file basesHumano/lncrna_pre.fasta --type AAC
```

[ ]:
```
!python iFeature/iFeature.py --file basesHumano/rna_presampling.fasta --type AAC
```

[ ]:
```
import pandas as pd
```

[ ]:
```
dflncRNA = pd.read_csv('basesHumano/AAC_mod_lncRNA.csv',sep=',')
```

[ ]:
```
dflncRNA.head()
```

[ ]:
```
dflncRNA['label'] = 'lncRNA'
```

[ ]:
```
display(dflncRNA)
```

[ ]:
```
dflncRNA.to_csv('AAC_lncRNA.csv',index=False,sep=',')
```

[ ]:
```
dfmRNA = pd.read_csv('basesHumano/AAC_mod_mRNA.csv',sep=',')
```

[ ]:
```
display(dfmRNA)
```

[ ]:
```
dfmRNA['label'] = 'mRNA'
```

[ ]:
```
display(dfmRNA)
```

[ ]:
```
dfmRNA.to_csv('AAC_mRNA.csv',index=False,sep=',')
```

0.4.6 Concatenando os datasets - iFeature + MathFeature (AAC + FOURIER + ORF)

[ ]:
```
%run MathFeature/preprocessing/concatenate.py -n 3 -o basesHumano/lncRNA.csv
```

[ ]:
```
%run MathFeature/preprocessing/concatenate.py -n 3 -o basesHumano/mRNA.csv
```

0.5 Divisão em treino e teste

[3]:
```
#importando as bibliotecas
import os
import pandas
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import cohen_kappa_score,confusion_matrix, accuracy_score, ␣
 ↪precision_score, recall_score, f1_score, roc_auc_score,roc_curve,auc,r2_score
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from imblearn.metrics import specificity_score
import matplotlib.pyplot  as pyplot
```

[ ]:
```
#Função para dividir em treino e teste
def split(finput, test_rate):
    dataset = pandas.read_csv(finput)
    X = dataset.iloc[:, :-1]
    y = dataset.iloc[:, -1]

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =␣
 ↪test_rate)
    train = pandas.concat([X_train, y_train], axis=1)
    test = pandas.concat([X_test, y_test], axis=1)

    trainData = os.path.splitext(finput)[0]+"_train"+os.path.splitext(finput)[1]
    testData = os.path.splitext(finput)[0]+"_test"+os.path.splitext(finput)[1]
    train.to_csv(trainData, index=False)
    test.to_csv(testData, index=False)
    return
```

[ ]:
```
# Aplica a divisão treino e teste nas bases mRNA e lncRNA
split('basesHumano/mRNA.csv',0.3)
split('basesHumano/lncRNA.csv',0.3)
```

[4]:
```
# carrega a base de dados treino lncRNA e mRNA
lncRNA_data = pandas.read_csv('basesHumano/lncRNA_train.csv')
```

```python
mRNA_data = pandas.read_csv('basesHumano/mRNA_train.csv')
dadosTreino = pandas.concat([lncRNA_data,mRNA_data])
```

[5]:
```python
dadosTreino.columns
```

[5]:
```
Index(['nameseq', 'A', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K', 'L', 'M', 'N',
       'P', 'Q', 'R', 'S', 'T', 'V', 'W', 'Y', 'average', 'median', 'maximum',
       'minimum', 'peak', 'none_levated_peak', 'sample_standard_deviation',
       'population_standard_deviation', 'percentile15', 'percentile25',
       'percentile50', 'percentile75', 'amplitude', 'variance',
       'interquartile_range', 'semi_interquartile_range',
       'coefficient_of_variation', 'skewness', 'kurtosis',
       'maximum_ORF_length', 'minimum_ORF_length', 'std_ORF_length',
       'average_ORF_length', 'cv_ORF_length', 'maximum_GC_content_ORF',
       'minimum_GC_content_ORF', 'std_GC_content_ORF',
       'average_GC_content_ORF', 'cv_GC_content_ORF', 'label'],
      dtype='object')
```

[6]:
```python
display(dadosTreino)
```

```
                                                  nameseq         A         C  \
0       ENST00000624128.2|ENSG00000203875.13|OTTHUMG00...  0.273009  0.199536
1       ENST00000657993.1|ENSG00000239523.6|OTTHUMG000...  0.323664  0.206107
2       ENST00000553454.1|ENSG00000215256.4|OTTHUMG000...  0.239946  0.266086
3       ENST00000528133.1|ENSG00000254676.1|OTTHUMG000...  0.264088  0.184530
4       ENST00000669022.1|ENSG00000275830.2|OTTHUMG000...  0.261066  0.239386
...                                                   ...       ...       ...
68106   ENST00000368092.7|ENSG00000162723.10|OTTHUMG00...  0.247350  0.270907
68107   ENST00000253099.11|ENSG00000105364.14|OTTHUMG0...  0.192939  0.330870
68108   ENST00000648544.1|ENSG00000164543.7|OTTHUMG000...  0.296316  0.207688
68109   ENST00000457054.6|ENSG00000170248.15|OTTHUMG00...  0.306105  0.192385
68110   ENST00000536594.5|ENSG00000173262.12|OTTHUMG00...  0.246777  0.265193

          D    E    F         G    H    I    K  ...  minimum_ORF_length  \
0       0.0  0.0  0.0  0.278422  0.0  0.0  0.0  ...                   9
1       0.0  0.0  0.0  0.225954  0.0  0.0  0.0  ...                   9
2       0.0  0.0  0.0  0.280161  0.0  0.0  0.0  ...                   9
3       0.0  0.0  0.0  0.205525  0.0  0.0  0.0  ...                   9
4       0.0  0.0  0.0  0.222222  0.0  0.0  0.0  ...                   6
...     ...  ...  ...       ...  ...  ...  ...  ...                 ...
68106   0.0  0.0  0.0  0.253239  0.0  0.0  0.0  ...                  18
68107   0.0  0.0  0.0  0.305419  0.0  0.0  0.0  ...                   6
68108   0.0  0.0  0.0  0.256273  0.0  0.0  0.0  ...                   6
68109   0.0  0.0  0.0  0.200604  0.0  0.0  0.0  ...                   6
68110   0.0  0.0  0.0  0.265193  0.0  0.0  0.0  ...                  21

       std_ORF_length  average_ORF_length  cv_ORF_length  \
0           48.826094           53.250000       0.916922
```

```
1          54.977814          67.200000       0.818122
2          61.747470          89.500000       0.689916
3          45.753688          48.000000       0.953202
4          34.125650          30.200000       1.129988
...              ...                ...            ...
68106     201.868769         149.000000       1.354824
68107     374.642830         307.500000       1.218351
68108     266.955784         125.368421       2.129370
68109     305.954888          85.356164       3.584450
68110      42.532341          72.000000       0.590727

       maximum_GC_content_ORF  minimum_GC_content_ORF  std_GC_content_ORF  \
0                   58.823529               11.111111           13.563062
1                   44.444444               31.428571            5.450845
2                   71.345029               38.095238            9.501427
3                   48.387097               22.222222            7.430612
4                   63.157895               16.666667           11.432097
...                       ...                     ...                 ...
68106               56.000000               38.888889            5.424358
68107               65.064103               33.333333           12.481002
68108               55.172414               20.833333            7.740153
68109               58.333333               13.333333            9.018525
68110               56.756757               42.857143            5.244292

       average_GC_content_ORF  cv_GC_content_ORF   label
0                   38.250925           0.354581  lncRNA
1                   38.166056           0.142819  lncRNA
2                   49.701641           0.191169  lncRNA
3                   30.658103           0.242370  lncRNA
4                   40.144412           0.284774  lncRNA
...                       ...                ...     ...
68106               49.936866           0.108624    mRNA
68107               54.264172           0.230004    mRNA
68108               38.693437           0.200038    mRNA
68109               32.991789           0.273357    mRNA
68110               51.612877           0.101608    mRNA

[136222 rows x 51 columns]
```

[7]:
```python
#Remove column nameseq
dadosTreino.drop(columns='nameseq', inplace=True)
```

[8]:
```python
#Vamos verificar
dadosTreino.columns
```

[8]:
```
Index(['A', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K', 'L', 'M', 'N', 'P', 'Q',
       'R', 'S', 'T', 'V', 'W', 'Y', 'average', 'median', 'maximum', 'minimum',
```

```
 'peak', 'none_levated_peak', 'sample_standard_deviation',
 'population_standard_deviation', 'percentile15', 'percentile25',
 'percentile50', 'percentile75', 'amplitude', 'variance',
 'interquartile_range', 'semi_interquartile_range',
 'coefficient_of_variation', 'skewness', 'kurtosis',
 'maximum_ORF_length', 'minimum_ORF_length', 'std_ORF_length',
 'average_ORF_length', 'cv_ORF_length', 'maximum_GC_content_ORF',
 'minimum_GC_content_ORF', 'std_GC_content_ORF',
 'average_GC_content_ORF', 'cv_GC_content_ORF', 'label'],
dtype='object')
```

[9]:
```
#Verificar valores nulos
dadosTreino.isnull().sum()
```

[9]:
```
A                                0
C                                0
D                                0
E                                0
F                                0
G                                0
H                                0
I                                0
K                                0
L                                0
M                                0
N                                0
P                                0
Q                                0
R                                0
S                                0
T                                0
V                                0
W                                0
Y                                0
average                          0
median                           0
maximum                          0
minimum                          0
peak                             0
none_levated_peak                0
sample_standard_deviation        0
population_standard_deviation    0
percentile15                     0
percentile25                     0
percentile50                     0
percentile75                     0
amplitude                        0
```

```
variance                        0
interquartile_range             0
semi_interquartile_range        0
coefficient_of_variation        0
skewness                        0
kurtosis                        0
maximum_ORF_length              0
minimum_ORF_length              0
std_ORF_length                  0
average_ORF_length              0
cv_ORF_length                   0
maximum_GC_content_ORF          0
minimum_GC_content_ORF          0
std_GC_content_ORF              0
average_GC_content_ORF          0
cv_GC_content_ORF               0
label                           0
dtype: int64
```

[10]:
```
#Fazendo uma cópia dos dados
dadosTreinoAux = dadosTreino.copy()
```

[11]:
```
display(dadosTreino)
```

```
              A         C    D    E    F         G    H    I    K    L  ...  \
0      0.273009  0.199536  0.0  0.0  0.0  0.278422  0.0  0.0  0.0  0.0  ...
1      0.323664  0.206107  0.0  0.0  0.0  0.225954  0.0  0.0  0.0  0.0  ...
2      0.239946  0.266086  0.0  0.0  0.0  0.280161  0.0  0.0  0.0  0.0  ...
3      0.264088  0.184530  0.0  0.0  0.0  0.205525  0.0  0.0  0.0  0.0  ...
4      0.261066  0.239386  0.0  0.0  0.0  0.222222  0.0  0.0  0.0  0.0  ...
...         ...       ...  ...  ...  ...       ...  ...  ...  ...  ...  ...
68106  0.247350  0.270907  0.0  0.0  0.0  0.253239  0.0  0.0  0.0  0.0  ...
68107  0.192939  0.330870  0.0  0.0  0.0  0.305419  0.0  0.0  0.0  0.0  ...
68108  0.296316  0.207688  0.0  0.0  0.0  0.256273  0.0  0.0  0.0  0.0  ...
68109  0.306105  0.192385  0.0  0.0  0.0  0.200604  0.0  0.0  0.0  0.0  ...
68110  0.246777  0.265193  0.0  0.0  0.0  0.265193  0.0  0.0  0.0  0.0  ...

       minimum_ORF_length  std_ORF_length  average_ORF_length  cv_ORF_length  \
0                       9       48.826094           53.250000       0.916922
1                       9       54.977814           67.200000       0.818122
2                       9       61.747470           89.500000       0.689916
3                       9       45.753688           48.000000       0.953202
4                       6       34.125650           30.200000       1.129988
...                   ...             ...                 ...            ...
68106                  18      201.868769          149.000000       1.354824
68107                   6      374.642830          307.500000       1.218351
68108                   6      266.955784          125.368421       2.129370
68109                   6      305.954888           85.356164       3.584450
```

```
68110                  21       42.532341          72.000000       0.590727

       maximum_GC_content_ORF  minimum_GC_content_ORF  std_GC_content_ORF  \
0                   58.823529               11.111111           13.563062
1                   44.444444               31.428571            5.450845
2                   71.345029               38.095238            9.501427
3                   48.387097               22.222222            7.430612
4                   63.157895               16.666667           11.432097
...                       ...                     ...                 ...
68106               56.000000               38.888889            5.424358
68107               65.064103               33.333333           12.481002
68108               55.172414               20.833333            7.740153
68109               58.333333               13.333333            9.018525
68110               56.756757               42.857143            5.244292

       average_GC_content_ORF  cv_GC_content_ORF   label
0                   38.250925           0.354581  lncRNA
1                   38.166056           0.142819  lncRNA
2                   49.701641           0.191169  lncRNA
3                   30.658103           0.242370  lncRNA
4                   40.144412           0.284774  lncRNA
...                       ...                ...     ...
68106               49.936866           0.108624    mRNA
68107               54.264172           0.230004    mRNA
68108               38.693437           0.200038    mRNA
68109               32.991789           0.273357    mRNA
68110               51.612877           0.101608    mRNA

[136222 rows x 50 columns]
```

0.6 Normalização dos dados treino

[12]:
```
#Transform categorical in binary class values
dicionario = {'mRNA':0,'lncRNA':1}
dadosTreino['label'] = dadosTreino['label'].map(dicionario)
```

[13]:
```
dadosTreino.columns
```

[13]:
```
Index(['A', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K', 'L', 'M', 'N', 'P', 'Q',
       'R', 'S', 'T', 'V', 'W', 'Y', 'average', 'median', 'maximum', 'minimum',
       'peak', 'none_levated_peak', 'sample_standard_deviation',
       'population_standard_deviation', 'percentile15', 'percentile25',
       'percentile50', 'percentile75', 'amplitude', 'variance',
       'interquartile_range', 'semi_interquartile_range',
       'coefficient_of_variation', 'skewness', 'kurtosis',
       'maximum_ORF_length', 'minimum_ORF_length', 'std_ORF_length',
       'average_ORF_length', 'cv_ORF_length', 'maximum_GC_content_ORF',
```

```
       'minimum_GC_content_ORF', 'std_GC_content_ORF',
       'average_GC_content_ORF', 'cv_GC_content_ORF', 'label'],
      dtype='object')
```

[14]:
```
#dadosTreino.iloc[:,20:49]
from sklearn.preprocessing import MinMaxScaler

# create a scaler object
scaler = MinMaxScaler()
# fit and transform the data
cols = dadosTreino.iloc[:, 20:49].columns
dadosTreino[cols] = pandas.DataFrame(scaler.fit_transform(dadosTreino.iloc[:, 20:
 ↪49]), columns=dadosTreino.iloc[:, 20:49].columns)
```

[15]:
```
dadosTreino
```

[15]:
```
              A         C    D    E    F         G    H    I    K    L  ...  \
0      0.273009  0.199536  0.0  0.0  0.0  0.278422  0.0  0.0  0.0  0.0  ...
1      0.323664  0.206107  0.0  0.0  0.0  0.225954  0.0  0.0  0.0  0.0  ...
2      0.239946  0.266086  0.0  0.0  0.0  0.280161  0.0  0.0  0.0  0.0  ...
3      0.264088  0.184530  0.0  0.0  0.0  0.205525  0.0  0.0  0.0  0.0  ...
4      0.261066  0.239386  0.0  0.0  0.0  0.222222  0.0  0.0  0.0  0.0  ...
...         ...       ...  ...  ...  ...       ...  ...  ...  ...  ...  ...
68106  0.247350  0.270907  0.0  0.0  0.0  0.253239  0.0  0.0  0.0  0.0  ...
68107  0.192939  0.330870  0.0  0.0  0.0  0.305419  0.0  0.0  0.0  0.0  ...
68108  0.296316  0.207688  0.0  0.0  0.0  0.256273  0.0  0.0  0.0  0.0  ...
68109  0.306105  0.192385  0.0  0.0  0.0  0.200604  0.0  0.0  0.0  0.0  ...
68110  0.246777  0.265193  0.0  0.0  0.0  0.265193  0.0  0.0  0.0  0.0  ...

       minimum_ORF_length  std_ORF_length  average_ORF_length  cv_ORF_length  \
0                0.006818        0.015199            0.039227       0.041929
1                0.006818        0.017114            0.049503       0.037411
2                0.006818        0.019222            0.065930       0.031549
3                0.006818        0.014243            0.035359       0.043588
4                0.004545        0.010623            0.022247       0.051673
...                   ...             ...                 ...            ...
68106            0.006818        0.010758            0.032747       0.035548
68107            0.006818        0.011110            0.038999       0.030827
68108            0.004545        0.010693            0.033763       0.034270
68109            0.006818        0.006282            0.018785       0.036189
68110            0.011364        0.034533            0.090608       0.041242

       maximum_GC_content_ORF  minimum_GC_content_ORF  std_GC_content_ORF  \
0                    0.641711                0.130952            0.474167
1                    0.484848                0.370408            0.190562
2                    0.778309                0.448980            0.332171
3                    0.527859                0.261905            0.259775
```

```
4                      0.688995                 0.196429               0.399668
...                         ...                      ...                    ...
68106                  0.648221                 0.235714               0.372699
68107                  0.539589                 0.245536               0.273451
68108                  0.631579                 0.196429               0.441372
68109                  0.606061                 0.471429               0.194890
68110                  0.694215                 0.336735               0.276514

       average_GC_content_ORF  cv_GC_content_ORF  label
0                      0.450814           0.493489      1
1                      0.449814           0.198769      1
2                      0.585769           0.266060      1
3                      0.361328           0.337319      1
4                      0.473131           0.396335      1
...                         ...                ...    ...
68106                  0.521885           0.335064      0
68107                  0.420313           0.305247      0
68108                  0.413964           0.500249      0
68109                  0.571280           0.160061      0
68110                  0.609109           0.212994      0

[136222 rows x 50 columns]
```

[16]:
```
#Divide a base entre os previsores e classe
colunas = dadosTreino.columns.drop('label')
```

[17]:
```
# Gera os previsores e classe (X e y)
X = dadosTreino[colunas].values
y = dadosTreino['label']
```

0.7 Dados de Teste

[18]:
```
# carrega a base de dados teste lncRNA e mRNA
lncRNA_data_t = pandas.read_csv('basesHumano/lncRNA_test.csv')
mRNA_data_t = pandas.read_csv('basesHumano/mRNA_test.csv')
dadosTeste = pandas.concat([lncRNA_data_t,mRNA_data_t])
```

[19]:
```
dadosTeste
```

[19]:
```
                                                 nameseq         A         C  \
0      ENST00000662662.1|ENSG00000255760.2|OTTHUMG000...  0.304718  0.249807
1      ENST00000670263.1|ENSG00000241472.7|OTTHUMG000...  0.296918  0.209130
2      ENST00000414989.2|ENSG00000224192.2|OTTHUMG000...  0.228037  0.261682
3      ENST00000656534.1|ENSG00000226995.9|OTTHUMG000...  0.239715  0.257120
4      ENST00000656913.1|ENSG00000267712.6|OTTHUMG000...  0.319322  0.205144
...                                                  ...       ...       ...
29186  ENST00000503281.6|ENSG00000164904.18|OTTHUMG00...  0.260406  0.214514
```

```
29187  ENST00000303645.10|ENSG00000170262.13|OTTHUMG0...  0.237634  0.310753
29188  ENST00000526322.5|ENSG00000149294.17|OTTHUMG00...  0.257143  0.269048
29189  ENST00000586262.5|ENSG00000091164.13|OTTHUMG00...  0.303869  0.166902
29190  ENST00000370952.4|ENSG00000066557.6|OTTHUMG000...  0.328878  0.166022

         D    E    F         G    H    I    K  ...  minimum_ORF_length  \
0      0.0  0.0  0.0  0.228925  0.0  0.0  0.0  ...                  18
1      0.0  0.0  0.0  0.196254  0.0  0.0  0.0  ...                   6
2      0.0  0.0  0.0  0.241121  0.0  0.0  0.0  ...                   6
3      0.0  0.0  0.0  0.265823  0.0  0.0  0.0  ...                   6
4      0.0  0.0  0.0  0.216437  0.0  0.0  0.0  ...                   6
...    ...  ...  ...       ...  ...  ...  ...  ...                 ...
29186  0.0  0.0  0.0  0.289221  0.0  0.0  0.0  ...                   9
29187  0.0  0.0  0.0  0.253763  0.0  0.0  0.0  ...                   6
29188  0.0  0.0  0.0  0.239683  0.0  0.0  0.0  ...                  21
29189  0.0  0.0  0.0  0.181870  0.0  0.0  0.0  ...                   6
29190  0.0  0.0  0.0  0.180795  0.0  0.0  0.0  ...                   6

       std_ORF_length  average_ORF_length  cv_ORF_length  \
0           81.694553           83.000000       0.984272
1           48.063540           47.581395       1.010133
2           41.173224           54.375000       0.757209
3           69.193641           67.800000       1.020555
4           58.135080           69.750000       0.833478
...               ...                 ...            ...
29186       21.330729           31.000000       0.688088
29187      162.172244          135.857143       1.193697
29188      166.349662          149.700000       1.111220
29189      118.819495           65.265306       1.820561
29190      295.637953           97.028571       3.046916

       maximum_GC_content_ORF  minimum_GC_content_ORF  std_GC_content_ORF  \
0                   62.500000               38.888889            6.006221
1                   57.692308                8.333333           11.198298
2                   58.333333               33.333333            7.625398
3                   60.000000               26.666667            9.253747
4                   61.538462               25.000000            7.593501
...                       ...                     ...                 ...
29186               60.416667               22.222222            9.840077
29187               64.341085               33.333333           10.030041
29188               60.185185               38.461538            7.148636
29189               47.222222                8.333333            8.372529
29190               55.555556                8.333333            9.593732

       average_GC_content_ORF  cv_GC_content_ORF   label
0                   48.185650           0.124648  lncRNA
1                   37.150870           0.301428  lncRNA
```

```
2                  46.705952           0.163264  lncRNA
3                  49.838720           0.185674  lncRNA
4                  40.751077           0.186339  lncRNA
...                      ...                ...     ...
29186              46.713802           0.210646    mRNA
29187              51.329949           0.195403    mRNA
29188              50.235307           0.142303    mRNA
29189              32.985070           0.253828    mRNA
29190              30.667726           0.312828    mRNA

[58382 rows x 51 columns]
```

[20]:
```python
#Remove column nameseq
dadosTeste.drop(columns='nameseq', inplace=True)
```

[21]:
```python
dadosTeste
```

[21]:
```
              A         C    D    E    F         G    H    I    K    L  ...  \
0      0.304718  0.249807  0.0  0.0  0.0  0.228925  0.0  0.0  0.0  0.0  ...
1      0.296918  0.209130  0.0  0.0  0.0  0.196254  0.0  0.0  0.0  0.0  ...
2      0.228037  0.261682  0.0  0.0  0.0  0.241121  0.0  0.0  0.0  0.0  ...
3      0.239715  0.257120  0.0  0.0  0.0  0.265823  0.0  0.0  0.0  0.0  ...
4      0.319322  0.205144  0.0  0.0  0.0  0.216437  0.0  0.0  0.0  0.0  ...
...         ...       ...  ...  ...  ...       ...  ...  ...  ...  ...  ...
29186  0.260406  0.214514  0.0  0.0  0.0  0.289221  0.0  0.0  0.0  0.0  ...
29187  0.237634  0.310753  0.0  0.0  0.0  0.253763  0.0  0.0  0.0  0.0  ...
29188  0.257143  0.269048  0.0  0.0  0.0  0.239683  0.0  0.0  0.0  0.0  ...
29189  0.303869  0.166902  0.0  0.0  0.0  0.181870  0.0  0.0  0.0  0.0  ...
29190  0.328878  0.166022  0.0  0.0  0.0  0.180795  0.0  0.0  0.0  0.0  ...

       minimum_ORF_length  std_ORF_length  average_ORF_length  cv_ORF_length  \
0                      18       81.694553           83.000000       0.984272
1                       6       48.063540           47.581395       1.010133
2                       6       41.173224           54.375000       0.757209
3                       6       69.193641           67.800000       1.020555
4                       6       58.135080           69.750000       0.833478
...                   ...             ...                 ...            ...
29186                   9       21.330729           31.000000       0.688088
29187                   6      162.172244          135.857143       1.193697
29188                  21      166.349662          149.700000       1.111220
29189                   6      118.819495           65.265306       1.820561
29190                   6      295.637953           97.028571       3.046916

       maximum_GC_content_ORF  minimum_GC_content_ORF  std_GC_content_ORF  \
0                   62.500000               38.888889            6.006221
1                   57.692308                8.333333           11.198298
2                   58.333333               33.333333            7.625398
```

```
3                 60.000000                 26.666667             9.253747
4                 61.538462                 25.000000             7.593501
...                     ...                       ...                  ...
29186             60.416667                 22.222222             9.840077
29187             64.341085                 33.333333            10.030041
29188             60.185185                 38.461538             7.148636
29189             47.222222                  8.333333             8.372529
29190             55.555556                  8.333333             9.593732

       average_GC_content_ORF  cv_GC_content_ORF   label
0                   48.185650           0.124648  lncRNA
1                   37.150870           0.301428  lncRNA
2                   46.705952           0.163264  lncRNA
3                   49.838720           0.185674  lncRNA
4                   40.751077           0.186339  lncRNA
...                       ...                ...     ...
29186               46.713802           0.210646    mRNA
29187               51.329949           0.195403    mRNA
29188               50.235307           0.142303    mRNA
29189               32.985070           0.253828    mRNA
29190               30.667726           0.312828    mRNA

[58382 rows x 50 columns]
```

[22]:

```
#Transform categorical in binary class values
dicionario = {'mRNA':0,'lncRNA':1}
dadosTeste['label'] = dadosTeste['label'].map(dicionario)
```

[23]:

```
dadosTeste
```

[23]:

```
              A         C    D    E    F         G    H    I    K    L  ...  \
0      0.304718  0.249807  0.0  0.0  0.0  0.228925  0.0  0.0  0.0  0.0  ...
1      0.296918  0.209130  0.0  0.0  0.0  0.196254  0.0  0.0  0.0  0.0  ...
2      0.228037  0.261682  0.0  0.0  0.0  0.241121  0.0  0.0  0.0  0.0  ...
3      0.239715  0.257120  0.0  0.0  0.0  0.265823  0.0  0.0  0.0  0.0  ...
4      0.319322  0.205144  0.0  0.0  0.0  0.216437  0.0  0.0  0.0  0.0  ...
...         ...       ...  ...  ...  ...       ...  ...  ...  ...  ...  ...
29186  0.260406  0.214514  0.0  0.0  0.0  0.289221  0.0  0.0  0.0  0.0  ...
29187  0.237634  0.310753  0.0  0.0  0.0  0.253763  0.0  0.0  0.0  0.0  ...
29188  0.257143  0.269048  0.0  0.0  0.0  0.239683  0.0  0.0  0.0  0.0  ...
29189  0.303869  0.166902  0.0  0.0  0.0  0.181870  0.0  0.0  0.0  0.0  ...
29190  0.328878  0.166022  0.0  0.0  0.0  0.180795  0.0  0.0  0.0  0.0  ...

       minimum_ORF_length  std_ORF_length  average_ORF_length  cv_ORF_length  \
0                      18       81.694553           83.000000       0.984272
1                       6       48.063540           47.581395       1.010133
2                       6       41.173224           54.375000       0.757209
```

```
3                          6   69.193641         67.800000     1.020555
4                          6   58.135080         69.750000     0.833478
...                      ...         ...               ...          ...
29186                      9   21.330729         31.000000     0.688088
29187                      6  162.172244        135.857143     1.193697
29188                     21  166.349662        149.700000     1.111220
29189                      6  118.819495         65.265306     1.820561
29190                      6  295.637953         97.028571     3.046916

       maximum_GC_content_ORF  minimum_GC_content_ORF  std_GC_content_ORF  \
0                   62.500000               38.888889            6.006221
1                   57.692308                8.333333           11.198298
2                   58.333333               33.333333            7.625398
3                   60.000000               26.666667            9.253747
4                   61.538462               25.000000            7.593501
...                       ...                     ...                 ...
29186               60.416667               22.222222            9.840077
29187               64.341085               33.333333           10.030041
29188               60.185185               38.461538            7.148636
29189               47.222222                8.333333            8.372529
29190               55.555556                8.333333            9.593732

       average_GC_content_ORF  cv_GC_content_ORF  label
0                   48.185650           0.124648      1
1                   37.150870           0.301428      1
2                   46.705952           0.163264      1
3                   49.838720           0.185674      1
4                   40.751077           0.186339      1
...                       ...                ...    ...
29186               46.713802           0.210646      0
29187               51.329949           0.195403      0
29188               50.235307           0.142303      0
29189               32.985070           0.253828      0
29190               30.667726           0.312828      0

[58382 rows x 50 columns]
```

0.8 Normalização dos dados Teste

[24]:
```python
scaler_t = MinMaxScaler()
# fit and transform the data
cols = dadosTeste.iloc[:, 20:49].columns
dadosTeste[cols] = pandas.DataFrame(scaler_t.fit_transform(dadosTeste.iloc[:, 20:
 ↪49]), columns=dadosTeste.iloc[:, 20:49].columns)
```

[25]:
```python
dadosTeste
```

[25]:

```
              A         C    D    E    F         G    H    I    K    L  ...  \
0      0.304718  0.249807  0.0  0.0  0.0  0.228925  0.0  0.0  0.0  0.0  ...
1      0.296918  0.209130  0.0  0.0  0.0  0.196254  0.0  0.0  0.0  0.0  ...
2      0.228037  0.261682  0.0  0.0  0.0  0.241121  0.0  0.0  0.0  0.0  ...
3      0.239715  0.257120  0.0  0.0  0.0  0.265823  0.0  0.0  0.0  0.0  ...
4      0.319322  0.205144  0.0  0.0  0.0  0.216437  0.0  0.0  0.0  0.0  ...
...         ...       ...  ...  ...  ...       ...  ...  ...  ...  ...  ...
29186  0.260406  0.214514  0.0  0.0  0.0  0.289221  0.0  0.0  0.0  0.0  ...
29187  0.237634  0.310753  0.0  0.0  0.0  0.253763  0.0  0.0  0.0  0.0  ...
29188  0.257143  0.269048  0.0  0.0  0.0  0.239683  0.0  0.0  0.0  0.0  ...
29189  0.303869  0.166902  0.0  0.0  0.0  0.181870  0.0  0.0  0.0  0.0  ...
29190  0.328878  0.166022  0.0  0.0  0.0  0.180795  0.0  0.0  0.0  0.0  ...

       minimum_ORF_length  std_ORF_length  average_ORF_length  cv_ORF_length  \
0                0.014118        0.025737            0.065098       0.045834
1                0.004706        0.015142            0.037319       0.047039
2                0.004706        0.012971            0.042647       0.035261
3                0.004706        0.021799            0.053176       0.047524
4                0.004706        0.018315            0.054706       0.038812
...                   ...             ...                 ...            ...
29186            0.004706        0.015951            0.052941       0.034930
29187            0.000000        0.000000            0.000000       0.000000
29188            0.011765        0.017307            0.041569       0.048268
29189            0.004706        0.010174            0.038503       0.030632
29190            0.004706        0.014468            0.041345       0.040567

       maximum_GC_content_ORF  minimum_GC_content_ORF  std_GC_content_ORF  \
0                    0.692935                0.458333            0.201771
1                    0.639632                0.098214            0.376193
2                    0.646739                0.392857            0.256166
3                    0.665217                0.314286            0.310868
4                    0.682274                0.294643            0.255094
...                       ...                     ...                 ...
29186                0.739130                0.392857            0.432938
29187                0.000000                0.000000            0.000000
29188                0.471196                0.261905            0.251306
29189                0.501553                0.261905            0.203249
29190                0.600543                0.196429            0.323362

       average_GC_content_ORF  cv_GC_content_ORF  label
0                    0.567902           0.171684      1
1                    0.437850           0.415174      1
2                    0.550463           0.224873      1
3                    0.587385           0.255739      1
4                    0.480281           0.256655      1
...                       ...                ...    ...
29186                0.560395           0.373315      0
```

```
29187                0.000000           0.000000      0
29188                0.397664           0.305374      0
29189                0.420243           0.233708      0
29190                0.455604           0.342962      0

[58382 rows x 50 columns]
```

[26]:
```python
# Gera os previsores e classe (X e y)
X_teste = dadosTeste[colunas].values
y_teste = dadosTeste['label']
```

[27]:
```python
print(X_teste)
```

```
[[0.30471771 0.24980665 0.         ... 0.20177149 0.56790231 0.1716843 ]
 [0.29691767 0.20912993 0.         ... 0.37619281 0.43784954 0.41517387]
 [0.22803738 0.26168224 0.         ... 0.25616571 0.55046301 0.22487297]
 ...
 [0.25714286 0.26904762 0.         ... 0.25130635 0.39766393 0.30537373]
 [0.30386896 0.16690201 0.         ... 0.20324946 0.42024313 0.23370779]
 [0.32887795 0.16602181 0.         ... 0.32336194 0.45560438 0.34296154]]
```

[28]:
```python
print(y_teste)
```

```
0        1
1        1
2        1
3        1
4        1
        ..
29186    0
29187    0
29188    0
29189    0
29190    0
Name: label, Length: 58382, dtype: int64
```

[29]:
```python
# Exibe a quantidade de atributos
print("Columns size >>> %d"%len(colunas))

# Exibe o nome dos atributos
print(dadosTreino.columns)
```

```
Columns size >>> 49
Index(['A', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K', 'L', 'M', 'N', 'P', 'Q',
       'R', 'S', 'T', 'V', 'W', 'Y', 'average', 'median', 'maximum', 'minimum',
       'peak', 'none_levated_peak', 'sample_standard_deviation',
       'population_standard_deviation', 'percentile15', 'percentile25',
       'percentile50', 'percentile75', 'amplitude', 'variance',
       'interquartile_range', 'semi_interquartile_range',
```

```
       'coefficient_of_variation', 'skewness', 'kurtosis',
       'maximum_ORF_length', 'minimum_ORF_length', 'std_ORF_length',
       'average_ORF_length', 'cv_ORF_length', 'maximum_GC_content_ORF',
       'minimum_GC_content_ORF', 'std_GC_content_ORF',
       'average_GC_content_ORF', 'cv_GC_content_ORF', 'label'],
      dtype='object')
```

[30]:
```
print(X.shape, y.shape, X_teste.shape, y_teste.shape)
```

```
(136222, 49) (136222,) (58382, 49) (58382,)
```

0.9 Aplica o modelo de predição com RandomForest sem o Feature Importance

[31]:
```
from sklearn.ensemble import RandomForestClassifier
# instancia um DecisionTreeClassifier
clf_rf = RandomForestClassifier(n_estimators = 10, criterion =␣
 ↪'entropy',random_state=123)
# treina o DT
clf_rf.fit(X, y)

y_pred = clf_rf.predict(X_teste)
#print(y_pred)

 # gerar score baseado na acurácia
acuracidade = round(accuracy_score(y_teste,y_pred)*100,2)
print(acuracidade)
```

```
69.04
```

0.10 Aplica o modelo de predição com RandomForest e Wrapper

[32]:
```
from sklearn.feature_selection import RFE

clf_rf_2 = RandomForestClassifier(n_estimators = 10, criterion =␣
 ↪'entropy',random_state=123)
rfe = RFE(estimator=clf_rf_2,n_features_to_select=10,step=1)
rfe = rfe.fit(X,y)
```

[33]:
```
#Armazena a nova dimensão do vetor de características
features = rfe.fit_transform(X,y)
```

[34]:
```
#Verifica a quantidade
print(features.shape)
```

```
(136222, 10)
```

0.11 Obtendo as 10 melhores features

[35]:
```
temp = pandas.Series(rfe.support_,index = colunas)
wrapperApproach = temp[temp==True].index
print(wrapperApproach)
```

```
Index(['A', 'C', 'G', 'T', 'peak', 'none_levated_peak', 'average_ORF_length',
       'maximum_GC_content_ORF', 'std_GC_content_ORF',
       'average_GC_content_ORF'],
      dtype='object')
```

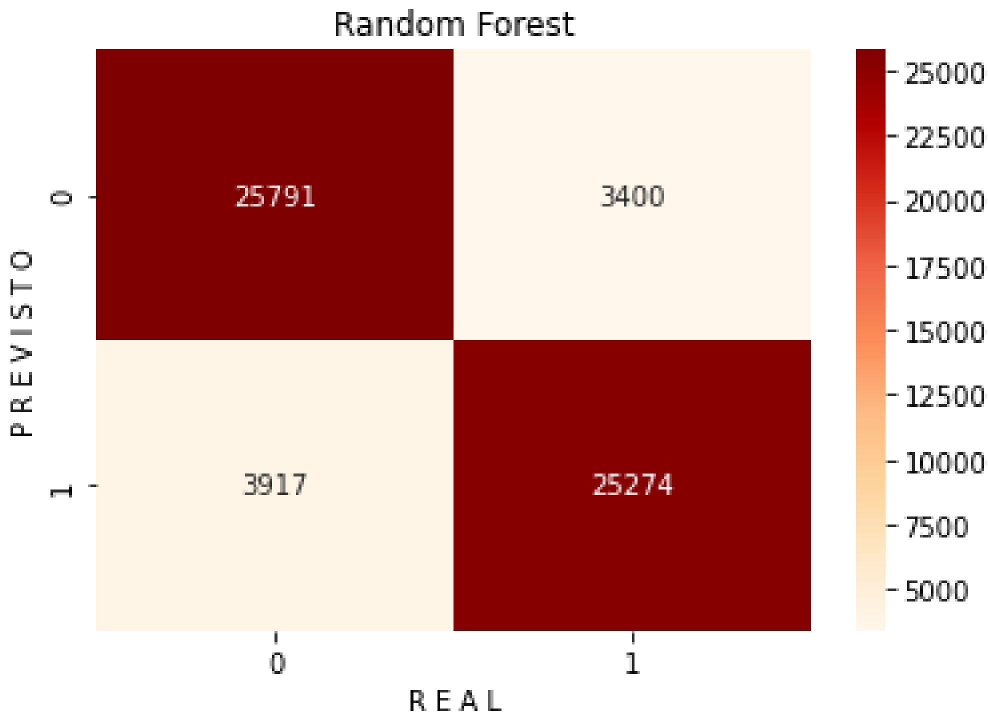
[36]:
```
#Predicao sem validação cruzada
y_pred = rfe.predict(X_teste)
acuracidade = round(accuracy_score(y_teste,y_pred)*100,2)
print(acuracidade)
```

```
87.47
```

[37]:
```
rf_pred = rfe.predict(X_teste)
```

0.12 Calculando as métricas

[38]:
```
#Matriz de confusão
from sklearn.metrics import confusion_matrix
import seaborn as sns
sns.heatmap(confusion_matrix(y_teste, rf_pred), cmap='OrRd', annot=True, fmt='2.
 ↪0f')
plt.title('Random Forest')
plt.ylabel('P R E V I S T O')
plt.xlabel('R E A L')
plt.show()
```



[39]:

```
#Acurácia, Sensibilidade positiva (VP/(VP+FN), Especificidade, Precisão, Recall,␣
↪F1-Score
acuracia_rf = accuracy_score(y_teste,rf_pred)
especificidade_rf = specificity_score(y_teste,rf_pred)
precisao_rf = precision_score(y_teste,rf_pred)
recall_rf = recall_score(y_teste,y_pred)
f1Score_rf = f1_score(y_teste,rf_pred)
curva_roc_escore_rf = roc_auc_score(y_teste,rf_pred)
kappa_rf = cohen_kappa_score(y_teste,rf_pred)
print(f'Acurácia:{round(acuracia_rf,2)}')
print(f'Especificidade:{round(especificidade_rf,2)}')
print(f'Precisão:{round(precisao_rf,2)}')
print(f'Recall ou Sensibilidade:{round(recall_rf,2)}')
print(f'F1-Score:{round(f1Score_rf,2)}')
print(f'Kappa:{round(kappa_rf,2)}')
print(f'Curva ROC:{round(curva_roc_escore_rf,2)}')
```

```
Acurácia:0.87
Especificidade:0.88
Precisão:0.88
Recall ou Sensibilidade:0.87
F1-Score:0.87
Kappa:0.75
Curva ROC:0.87
```

0.13 Aplica o modelo de predição com Regressão Logística e Wrapper

[40]:
```
from sklearn.linear_model import LogisticRegression
clf_rl = LogisticRegression(max_iter=2000)
rfe_rl = RFE(clf_rl,n_features_to_select=10,step=1)
fit_rl = rfe_rl.fit(X,y)
```

[41]:
```
#Armazena a nova dimensão do vetor de características
features_rl = fit_rl.fit_transform(X,y)
```

[42]:
```
#Verifica a quantidade
print(features_rl.shape)
```

```
(136222, 10)
```

0.14 Exibindo as 10 melhores features

[43]:
```
temp_rl = pandas.Series(fit_rl.support_,index = colunas)
wrapperApproach_rl = temp_rl[temp_rl==True].index
print(wrapperApproach_rl)
```

```
Index(['A', 'C', 'T', 'none_levated_peak', 'kurtosis',
       'maximum_GC_content_ORF', 'minimum_GC_content_ORF',
       'std_GC_content_ORF', 'average_GC_content_ORF', 'cv_GC_content_ORF'],
      dtype='object')
```

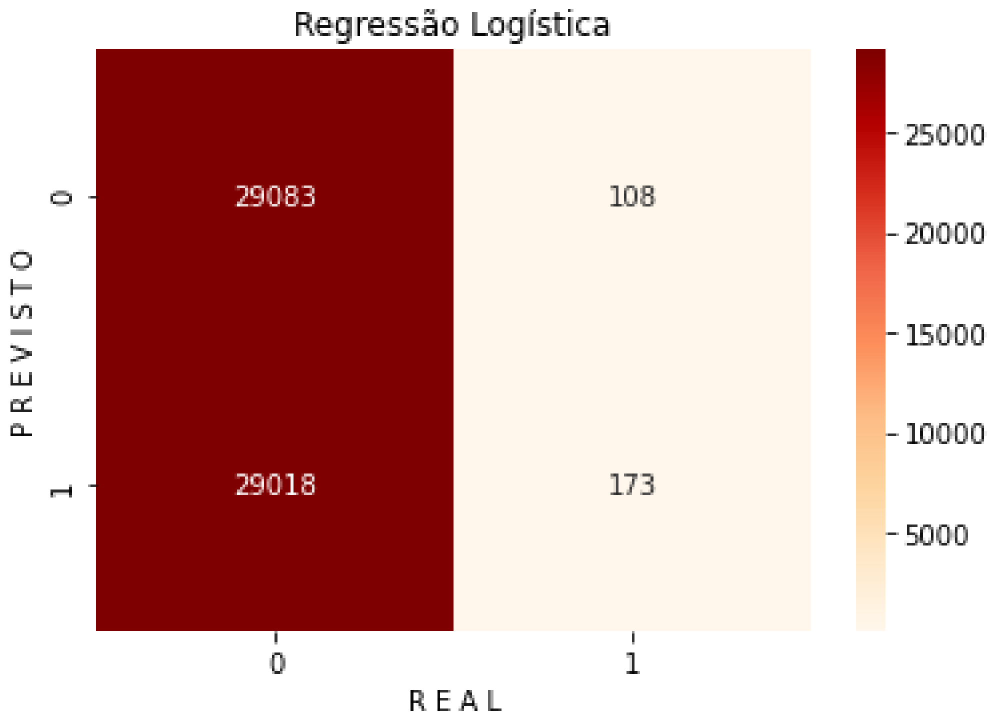
[44]:
```
#Predicao sem validação cruzada
y_pred_rl = fit_rl.predict(X_teste)
acuracidade_rl = round(accuracy_score(y_teste,y_pred_rl)*100,2)
print(acuracidade_rl)
```

```
50.11
```

[45]:
```
rl_pred = fit_rl.predict(X_teste)
```

0.15 Calculando as métricas

[46]:
```
#Matriz de confusão
sns.heatmap(confusion_matrix(y_teste, rl_pred), cmap='OrRd', annot=True, fmt='2.
 ↪0f')
plt.title('Regressão Logística')
plt.ylabel('P R E V I S T O')
plt.xlabel('R E A L')
plt.show()
```

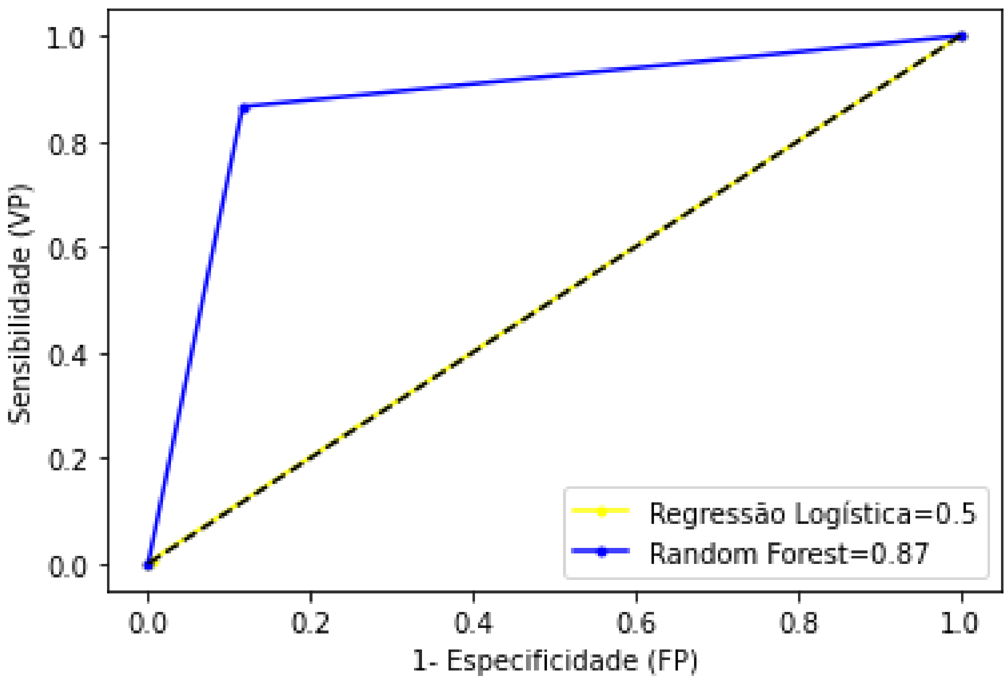


[47]:
```
#Acurácia, Sensibilidade positiva (VP/(VP+FN), Especificidade, Precisão, Recall,␣
↪F1-Score
acuracia_rl = accuracy_score(y_teste,rl_pred)
especificidade_rl = specificity_score(y_teste,rl_pred)
precisao_rl = precision_score(y_teste,rl_pred)
recall_rl = recall_score(y_teste,rl_pred)
f1Score_rl = f1_score(y_teste,rl_pred)
curva_roc_escore_rl = roc_auc_score(y_teste,rl_pred)
kappa_rl = cohen_kappa_score(y_teste,rl_pred)
print(f'Acurácia:{round(acuracia_rl,2)}')
print(f'Especificidade:{round(especificidade_rl,2)}')
print(f'Precisão:{round(precisao_rl,2)}')
print(f'Recall ou Sensibilidade:{round(recall_rl,2)}')
print(f'F1-Score:{round(f1Score_rl,2)}')
print(f'Kappa:{round(kappa_rl,2)}')
print(f'Curva ROC:{round(curva_roc_escore_rl,2)}')
```

```
Acurácia:0.5
Especificidade:1.0
Precisão:0.62
Recall ou Sensibilidade:0.01
F1-Score:0.01
Kappa:0.0
Curva ROC:0.5
```

0.16 Curva ROC

[48]:
```
rfp_rl, rvp_rl,lim1 = roc_curve(y_teste,rl_pred)
rfp_rf, rvp_rf,lim2 = roc_curve(y_teste,rf_pred)
pyplot.plot(rfp_rl, rvp_rl, marker='.', label='Regressão␣
↪Logística='+str(round(curva_roc_escore_rl,2)),color='yellow')
pyplot.plot(rfp_rf, rvp_rf, marker='.', label='Random␣
↪Forest='+str(round(curva_roc_escore_rf,2)),color='blue')
pyplot.plot([0, 1], [0, 1], color='black', linestyle='--')
# alterando o nome dos eixos
pyplot.xlabel('1- Especificidade (FP)')
pyplot.ylabel('Sensibilidade (VP)')
pyplot.legend()
# Mostrando o gráfico
pyplot.show()
```

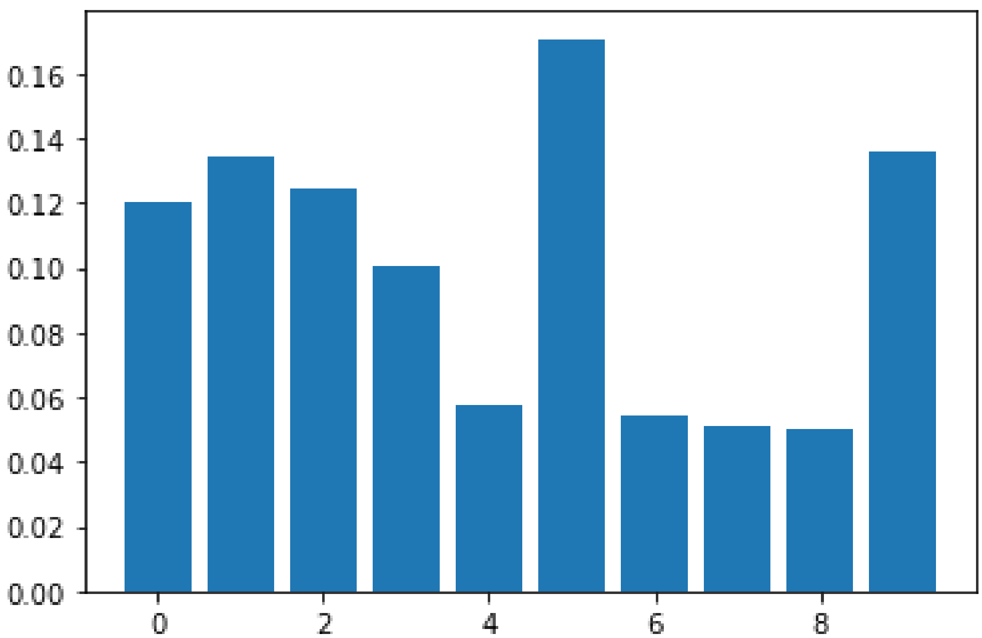


0.17 Análise por Feature Importance (Método Wrapper)

[49]:
```
# decision tree for feature importance on a regression problem
# define the model
featuresList = wrapperApproach.tolist()
model = RandomForestClassifier(n_estimators = 10, criterion =␣
↪'entropy',random_state=43)
# fit the model
```

```
model.fit(features, y)
# get importance
importance = model.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
    print('Feature %s - score %.5f' % (featuresList[i], v) )
    #print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```

```
Feature A - score 0.12039
Feature C - score 0.13420
Feature G - score 0.12441
Feature T - score 0.10071
Feature peak - score 0.05740
Feature none_levated_peak - score 0.17094
Feature average_ORF_length - score 0.05462
Feature maximum_GC_content_ORF - score 0.05104
Feature std_GC_content_ORF - score 0.05054
Feature average_GC_content_ORF - score 0.13576
```



0.18 Aplica o modelo de predição com RandomForest e Filtro

[50]:
```
# Import the necessary libraries first
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import mutual_info_classif
selector = SelectKBest(score_func=mutual_info_classif, k=10)
selector.fit(X, y)
```

[50]:
```
SelectKBest(score_func=<function mutual_info_classif at 0x7ff151785a60>)
```

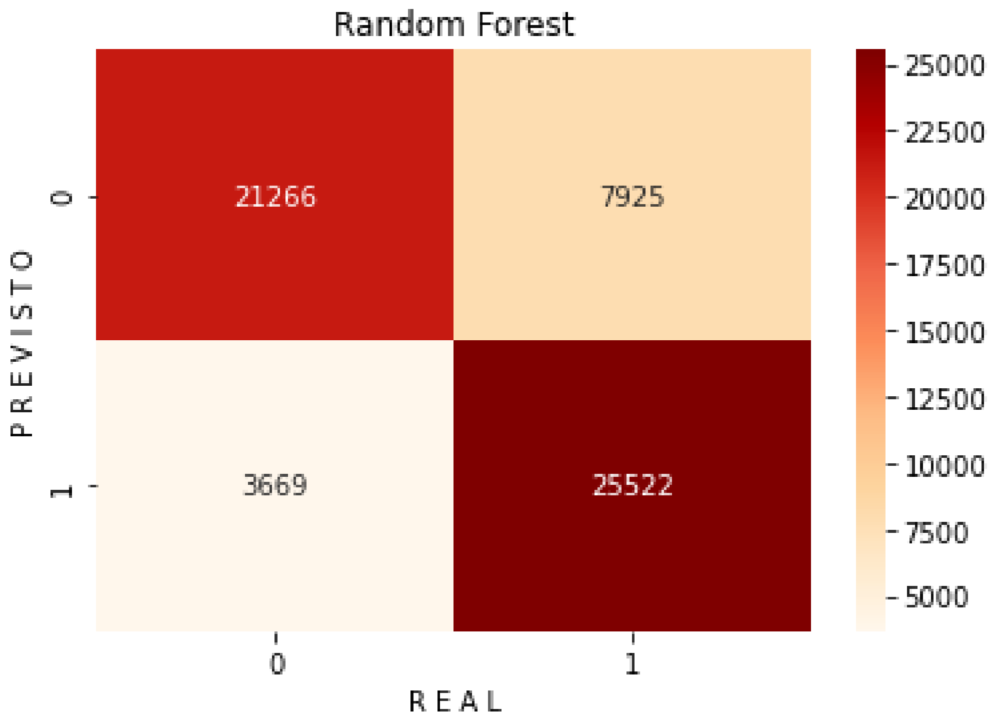
[51]:
```
# to remove the rest of the features:
X_train_filtro = selector.transform(X)
X_teste_filtro = selector.transform(X_teste)
```

[52]:
```
#Executando o modelo
clf_rf_filtro = RandomForestClassifier(random_state=123)
clr_rf_filtro = clf_rf_filtro.fit(X_train_filtro,y)
```

[53]:
```
#Predição
rf_pred_filtro = clr_rf_filtro.predict(X_teste_filtro)
```

0.19 Calculando as métricas

[54]:
```
#Matriz de confusão
sns.heatmap(confusion_matrix(y_teste, rf_pred_filtro), cmap='OrRd', annot=True,␣
 ↪fmt='2.0f')
plt.title('Random Forest')
plt.ylabel('P R E V I S T O')
plt.xlabel('R E A L')
plt.show()
```



[55]:

```
#Acurácia, Sensibilidade positiva (VP/(VP+FN), Especificidade, Precisão, Recall,␣
↪F1-Score
acuracia_rf_f = accuracy_score(y_teste,rf_pred_filtro)
especificidade_rf_f = specificity_score(y_teste,rf_pred_filtro)
precisao_rf_f = precision_score(y_teste,rf_pred_filtro)
recall_rf_f = recall_score(y_teste,rf_pred_filtro)
f1Score_rf_f = f1_score(y_teste,rf_pred_filtro)
curva_roc_escore_rf_f = roc_auc_score(y_teste,rf_pred_filtro)
kappa_rf_f = cohen_kappa_score(y_teste,rf_pred_filtro)
print(f'Acurácia:{round(acuracia_rf_f,2)}')
print(f'Especificidade:{round(especificidade_rf_f,2)}')
print(f'Precisão:{round(precisao_rf_f,2)}')
print(f'Recall ou Sensibilidade:{round(recall_rf_f,2)}')
print(f'F1-Score:{round(f1Score_rf_f,2)}')
print(f'Kappa:{round(kappa_rf_f,2)}')
print(f'Curva ROC:{round(curva_roc_escore_rf_f,2)}')
```

```
Acurácia:0.8
Especificidade:0.73
Precisão:0.76
Recall ou Sensibilidade:0.87
F1-Score:0.81
Kappa:0.6
Curva ROC:0.8
```

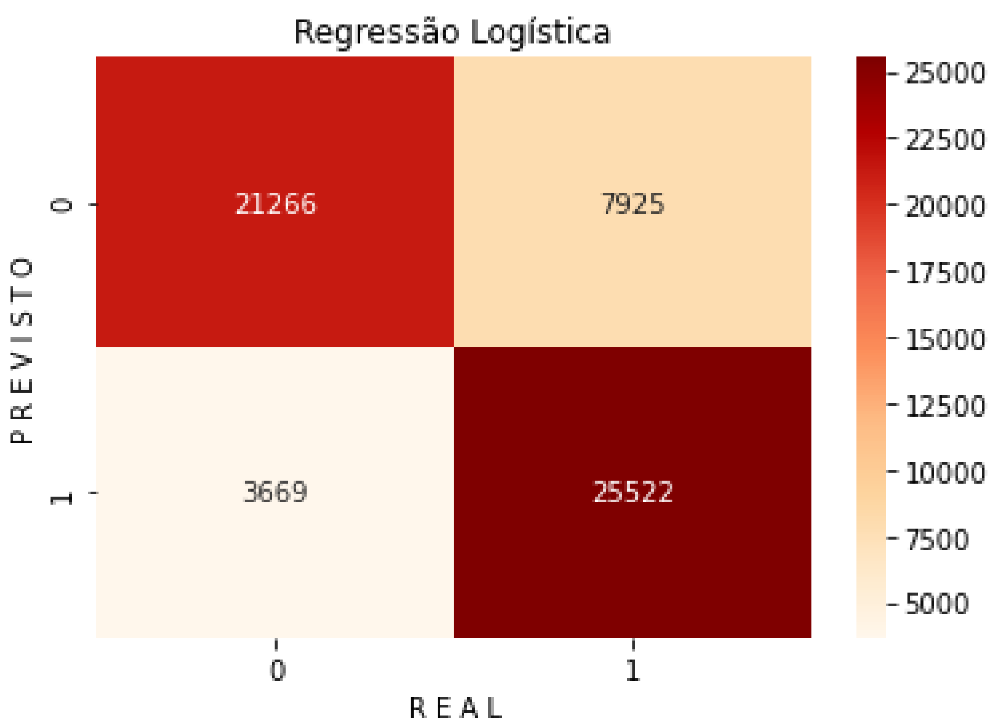
0.20 Aplica o modelo de predição com Regressão Logística e Filter

[56]:
```
#Executando o modelo
clf_rl_filtro = LogisticRegression(max_iter=2000)
clr_rl_filtro = clf_rf_filtro.fit(X_train_filtro,y)
```

[57]:
```
#Predição
rl_pred_filtro = clr_rl_filtro.predict(X_teste_filtro)
```

0.21 Calculando as métricas

[58]:
```
#Matriz de confusão
sns.heatmap(confusion_matrix(y_teste, rl_pred_filtro), cmap='OrRd', annot=True,␣
 ↪fmt='2.0f')
plt.title('Regressão Logística')
plt.ylabel('P R E V I S T O')
plt.xlabel('R E A L')
plt.show()
```



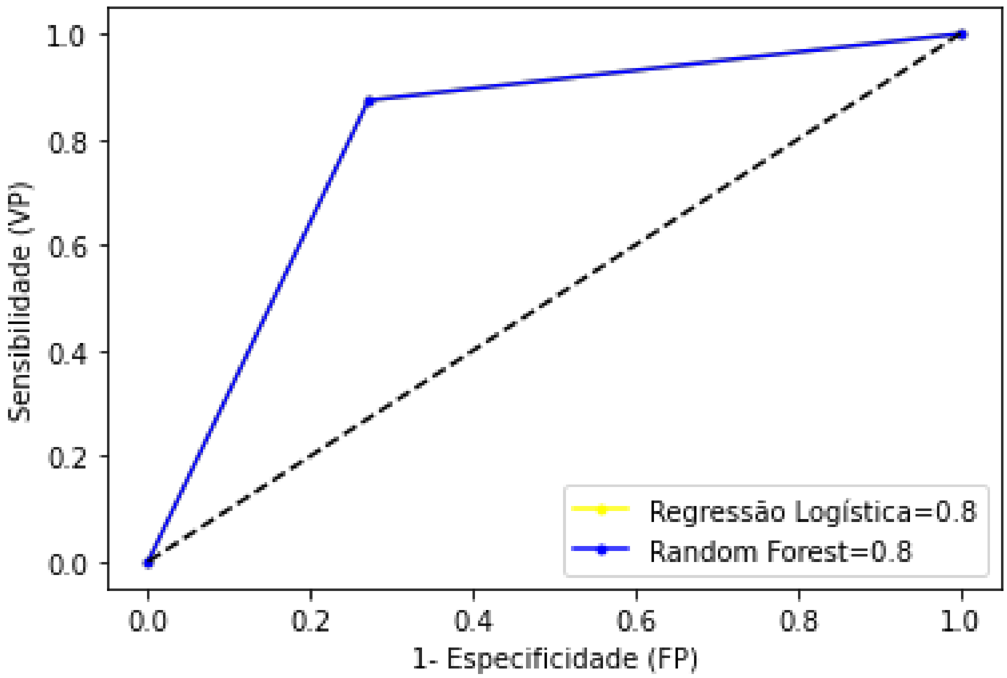
[59]:
```
#Acurácia, Sensibilidade positiva (VP/(VP+FN), Especificidade, Precisão, Recall,␣
 ↪F1-Score
acuracia_rl_f = accuracy_score(y_teste,rl_pred_filtro)
especificidade_rl_f = specificity_score(y_teste,rl_pred_filtro)
```

```
precisao_rl_f = precision_score(y_teste,rl_pred_filtro)
recall_rl_f = recall_score(y_teste,rl_pred_filtro)
f1Score_rl_f = f1_score(y_teste,rl_pred_filtro)
curva_roc_escore_rl_f = roc_auc_score(y_teste,rl_pred_filtro)
kappa_rl_f = cohen_kappa_score(y_teste,rl_pred_filtro)
print(f'Acurácia:{round(acuracia_rl_f,2)}')
print(f'Especificidade:{round(especificidade_rl_f,2)}')
print(f'Precisão:{round(precisao_rf_f,2)}')
print(f'Recall ou Sensibilidade:{round(recall_rl_f,2)}')
print(f'F1-Score:{round(f1Score_rl_f,2)}')
print(f'Kappa:{round(kappa_rl_f,2)}')
print(f'Curva ROC:{round(curva_roc_escore_rl_f,2)}')
```

```
Acurácia:0.8
Especificidade:0.73
Precisão:0.76
Recall ou Sensibilidade:0.87
F1-Score:0.81
Kappa:0.6
Curva ROC:0.8
```

0.22 Curva ROC

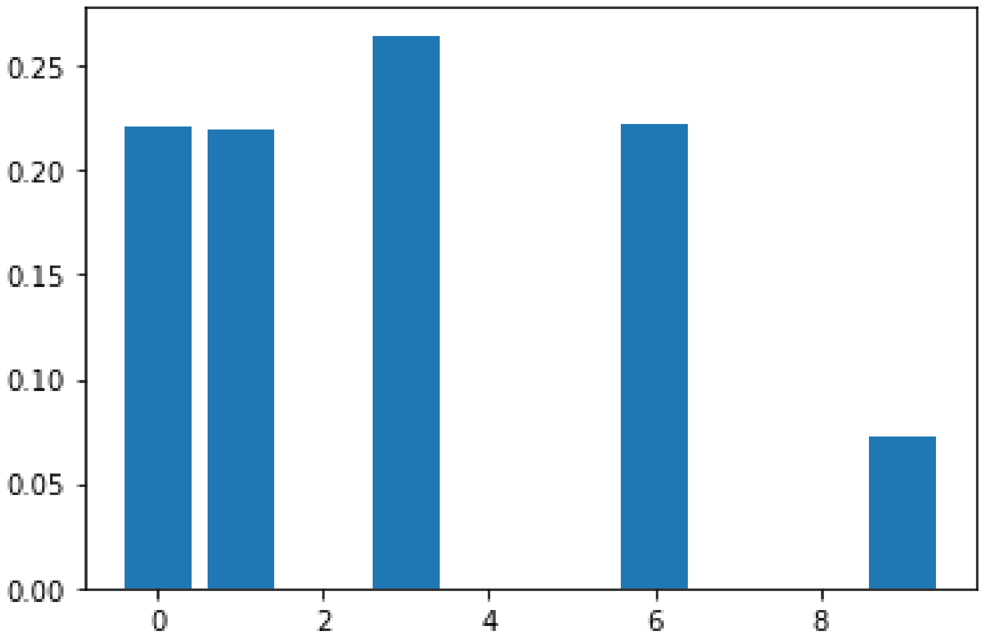
[67]:
```
rfp_rl_f, rvp_rl_f,lim4 = roc_curve(y_teste,rl_pred_filtro)
rfp_rf_f, rvp_rf_f,lim5 = roc_curve(y_teste,rf_pred_filtro)
pyplot.plot(rfp_rl_f, rvp_rl_f, marker='.', label='Regressão␣
↪Logística='+str(round(curva_roc_escore_rl_f,2)),color='yellow')
pyplot.plot(rfp_rf_f, rvp_rf_f, marker='.', label='Random␣
↪Forest='+str(round(curva_roc_escore_rf_f,2)),color='blue')
pyplot.plot([0, 1], [0, 1], color='black', linestyle='--')
# alterando o nome dos eixos
pyplot.xlabel('1- Especificidade (FP)')
pyplot.ylabel('Sensibilidade (VP)')
pyplot.legend()
# Mostrando o gráfico
pyplot.show()
```



0.23 Análise por Feature Importance (Método Filtro)

[71]:
```
colNames = dadosTreino.columns.tolist()
```

[73]:
```
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier
from matplotlib import pyplot
# define the model
model = RandomForestClassifier()
# fit the model
model.fit(X_train_filtro, y)
# get importance
importance = model.feature_importances_
# summarize feature importance
#for i,v in enumerate(importance):
 #    print('Feature %s - score %.5f' % (colNames[cols[i]], v) )
     #print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```



0.25
0.20
0.15
0.10
0.05
0.00
0
2
4
6
8