ProjetoMineracaoDados

December 7, 2021

[1]:
```
from IPython.display import Image
```

1 Projeto Final - Disciplina de Mineração da Dados

2 Autoencoder vs Filter Methods vs Wrapper Methods

3 Aluno: José Luiz Vilas Boas

4 Professor: Dr. Danilo Sipoli Sanches

4.0.1 Artigo: Prediction and prioritization of autism-associated long non-coding RNAs using gene expression and sequence features.

4.0.2 Autores: Wang, Jun e Wang, Liangjiang.

4.1 Objetivos do trabalho dos autores:

- Indentificar genes canditados ao Autism spectrum disorders (ASD);
- Desenvolveram um modelo de máquina machine learning para previsão e priorização de lncRNAs candidatos associados a ASD;
- Redução da dimensionalidade.

5 Autoencoders

- “Autoencoder é um tipo de rede neural que pode ser usada para aprender uma representação compactada de dados brutos” [1];
- “É um método de aprendizagem não supervisionado, embora, tecnicamente, sejam treinados por meio de métodos de aprendizagem supervisionados, denominados de auto-supervisão” [1].

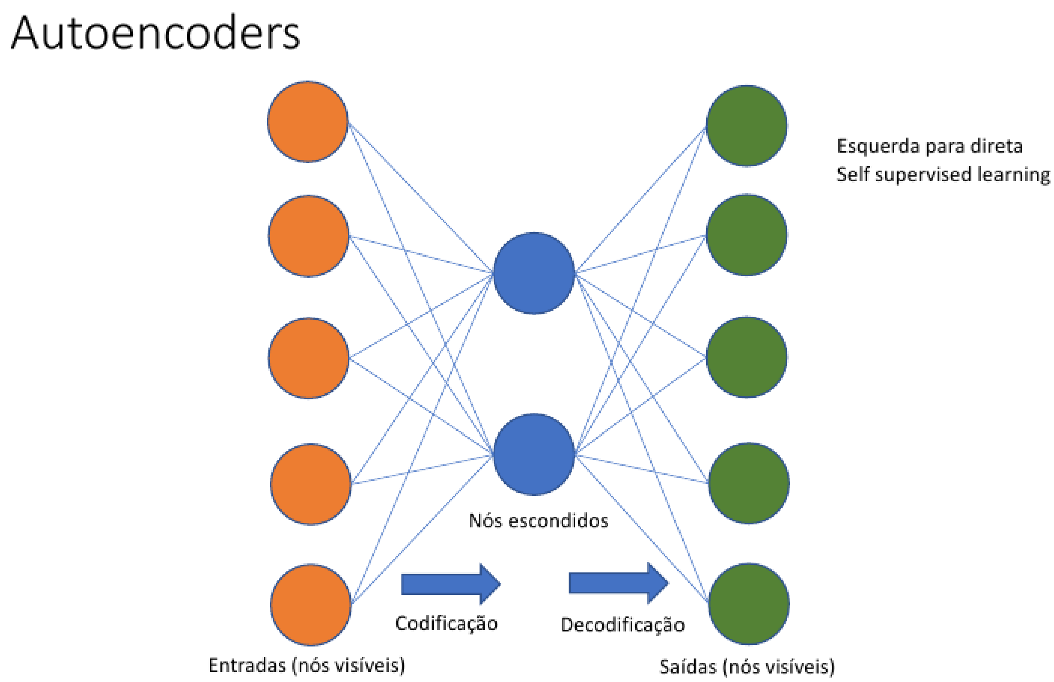
5.1 Comando para deixar iopub.data_rate maior que o padrão:

5.1.1 1 - Abra um jupyter notebook com o comando abaixo:

5.1.2 jupyter notebook –NotebookApp.iopub_data_rate_limit=1.0e10

[2]:
```
Image(filename='autoencoder.png')
```

[2]:



6 Materiais e ferramantas utilizadas

- Sequencias de lncRNA e RNA transcritos de humanos do repositório GENCODE: https://www.gencodegenes.org/;
- MathFeature e IFeature para Feature extraction;
- Seqkit para algumas funções de pré-processamento;
- Máquinas preditivas: Regressão Logística Random Forest para construção do modelo.

[4]:
```
Image(filename='metodologia.png')
```

[4]:

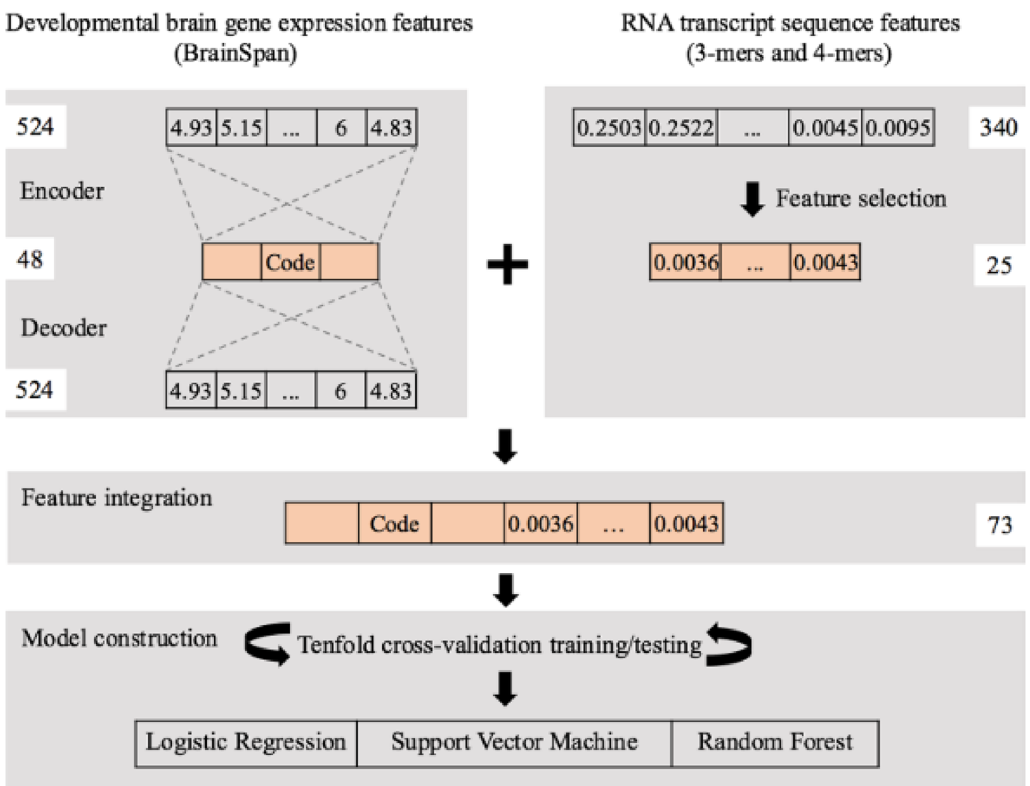


Fig. 1 Schematic diagram of model construction. The autoencoder-derived expression features and selected sequence features (*k*-mers) were combined for model construction

6.1 Pré-processamento

6.1.1 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
```

6.1.2 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
```

6.2 Usando as funções de pré-processamento do MathFeature

6.2.1 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
```

6.2.2 Recontanto as sequências

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

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

6.2.3 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
```

6.2.4 Recontanto as sequências

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

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

6.3 Extração de características

6.3.1 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
```

6.3.2 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
```

6.3.3 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
```

6.3.4 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
```

6.3.5 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=',')
```

6.3.6 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
```

6.4 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
from sklearn.linear_model import LogisticRegression
```

[ ]:
```
#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)
```

[111]:
```
# carrega a base de dados treino lncRNA e mRNA
lncRNA_data = pandas.read_csv('basesHumano/lncRNA_train.csv')
mRNA_data = pandas.read_csv('basesHumano/mRNA_train.csv')
dadosTreino = pandas.concat([lncRNA_data,mRNA_data])
```

6.5 Redução da amostragem para 90%

[112]:
```
dadosTreino.shape
```

[112]:
```
(136222, 51)
```

[113]:
```
dadosTreino = dadosTreino.sample(frac = 0.90)
```

[114]:
```
dadosTreino.shape
```

[114]:
```
(122600, 51)
```

[115]:
```
dadosTreino.columns
```

[115]:
```
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')
```

[116]:
```
display(dadosTreino)
```

```
                                                nameseq         A         C  \
54832  ENST00000376099.5|ENSG00000204482.11|OTTHUMG00...  0.265574  0.286885
29225  ENST00000512693.1|ENSG00000249476.2|OTTHUMG000...  0.280387  0.230488
17358  ENST00000667427.1|ENSG00000258168.6|OTTHUMG000...  0.281202  0.216487
31924  ENST00000539826.6|ENSG00000100852.13|OTTHUMG00...  0.344602  0.179267
64165  ENST00000438813.1|ENSG00000161594.7|OTTHUMG000...  0.257951  0.249117
```

```
...                                                       ...       ...       ...
23308  ENST00000366376.3|ENSG00000203565.3|OTTHUMG000...  0.288520  0.204431
5578   ENST00000655683.1|ENSG00000253369.2|OTTHUMG000...  0.274664  0.226457
51197  ENST00000658782.1|ENSG00000242512.9|OTTHUMG000...  0.266002  0.211416
21805  ENST00000646400.1|ENSG00000237505.8|OTTHUMG000...  0.347059  0.179832
51734  ENST00000453965.2|ENSG00000234308.3|OTTHUMG000...  0.320988  0.204938

         D    E    F         G    H    I    K  ...  minimum_ORF_length  \
54832  0.0  0.0  0.0  0.232787  0.0  0.0  0.0  ...                  21
29225  0.0  0.0  0.0  0.215683  0.0  0.0  0.0  ...                   6
17358  0.0  0.0  0.0  0.228814  0.0  0.0  0.0  ...                   6
31924  0.0  0.0  0.0  0.198730  0.0  0.0  0.0  ...                   6
64165  0.0  0.0  0.0  0.265018  0.0  0.0  0.0  ...                   6
...    ...  ...  ...       ...  ...  ...  ...  ...                 ...
23308  0.0  0.0  0.0  0.197885  0.0  0.0  0.0  ...                  12
5578   0.0  0.0  0.0  0.230381  0.0  0.0  0.0  ...                   6
51197  0.0  0.0  0.0  0.234968  0.0  0.0  0.0  ...                   6
21805  0.0  0.0  0.0  0.204202  0.0  0.0  0.0  ...                   6
51734  0.0  0.0  0.0  0.214815  0.0  0.0  0.0  ...                   6

       std_ORF_length  average_ORF_length  cv_ORF_length  \
54832       55.099909          114.000000       0.483333
29225       60.921032           66.053571       0.922297
17358       50.138769           56.470588       0.887874
31924      583.561132          116.482759       5.009850
64165       72.677369          100.000000       0.726774
...               ...                 ...            ...
23308       71.321044           95.444444       0.747252
5578        42.273653           68.222222       0.619646
51197       50.940106           60.840000       0.837280
21805       34.403963           41.714286       0.824753
51734       41.934351           58.714286       0.714210

       maximum_GC_content_ORF  minimum_GC_content_ORF  std_GC_content_ORF  \
54832               59.895833               38.095238            7.921621
29225               63.333333               19.444444           10.944267
17358               58.333333               33.333333            7.671121
31924               52.380952               22.222222            7.018167
64165               50.273224               33.333333            6.920349
...                       ...                     ...                 ...
23308               54.545455               16.666667            9.923169
5578                56.589147               25.000000            9.698144
51197               58.333333               16.666667            8.918680
21805               52.380952               16.666667            8.039676
51734               52.222222               33.333333            6.603660

       average_GC_content_ORF  cv_GC_content_ORF  label
54832               52.281948           0.151517   mRNA
```

```
29225                 41.489933           0.263781  lncRNA
17358                 41.533255           0.184698  lncRNA
31924                 35.941962           0.195264    mRNA
64165                 41.982967           0.164837    mRNA
...                         ...                ...     ...
23308                 38.836177           0.255514  lncRNA
5578                  40.080033           0.241969  lncRNA
51197                 40.633165           0.219493  lncRNA
21805                 35.659156           0.225459  lncRNA
51734                 44.510547           0.148362  lncRNA

[122600 rows x 51 columns]
```

[117]:
```python
#Remove column nameseq
dadosTreino.drop(columns='nameseq', inplace=True)
```

[118]:
```python
#Vamos verificar
dadosTreino.columns
```

[118]:
```
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')
```

[119]:
```python
#Verificar valores nulos
dadosTreino.isnull().sum()
```

[119]:
```
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
```

[120]:
```
#Fazendo uma cópia dos dados
dadosTreinoAux = dadosTreino.copy()
```

[121]:
```
display(dadosTreino)
```

```
              A         C    D    E    F         G    H    I    K    L  ...  \
54832  0.265574  0.286885  0.0  0.0  0.0  0.232787  0.0  0.0  0.0  0.0  ...
29225  0.280387  0.230488  0.0  0.0  0.0  0.215683  0.0  0.0  0.0  0.0  ...
17358  0.281202  0.216487  0.0  0.0  0.0  0.228814  0.0  0.0  0.0  0.0  ...
```

```
31924  0.344602  0.179267  0.0  0.0  0.0  0.198730  0.0  0.0  0.0  0.0  ...
64165  0.257951  0.249117  0.0  0.0  0.0  0.265018  0.0  0.0  0.0  0.0  ...
...         ...       ...  ...  ...  ...       ...  ...  ...  ...  ...  ...
23308  0.288520  0.204431  0.0  0.0  0.0  0.197885  0.0  0.0  0.0  0.0  ...
5578   0.274664  0.226457  0.0  0.0  0.0  0.230381  0.0  0.0  0.0  0.0  ...
51197  0.266002  0.211416  0.0  0.0  0.0  0.234968  0.0  0.0  0.0  0.0  ...
21805  0.347059  0.179832  0.0  0.0  0.0  0.204202  0.0  0.0  0.0  0.0  ...
51734  0.320988  0.204938  0.0  0.0  0.0  0.214815  0.0  0.0  0.0  0.0  ...

       minimum_ORF_length  std_ORF_length  average_ORF_length  cv_ORF_length  \
54832                  21       55.099909          114.000000       0.483333
29225                   6       60.921032           66.053571       0.922297
17358                   6       50.138769           56.470588       0.887874
31924                   6      583.561132          116.482759       5.009850
64165                   6       72.677369          100.000000       0.726774
...                   ...             ...                 ...            ...
23308                  12       71.321044           95.444444       0.747252
5578                    6       42.273653           68.222222       0.619646
51197                   6       50.940106           60.840000       0.837280
21805                   6       34.403963           41.714286       0.824753
51734                   6       41.934351           58.714286       0.714210

       maximum_GC_content_ORF  minimum_GC_content_ORF  std_GC_content_ORF  \
54832               59.895833               38.095238            7.921621
29225               63.333333               19.444444           10.944267
17358               58.333333               33.333333            7.671121
31924               52.380952               22.222222            7.018167
64165               50.273224               33.333333            6.920349
...                       ...                     ...                 ...
23308               54.545455               16.666667            9.923169
5578                56.589147               25.000000            9.698144
51197               58.333333               16.666667            8.918680
21805               52.380952               16.666667            8.039676
51734               52.222222               33.333333            6.603660

       average_GC_content_ORF  cv_GC_content_ORF   label
54832               52.281948           0.151517    mRNA
29225               41.489933           0.263781  lncRNA
17358               41.533255           0.184698  lncRNA
31924               35.941962           0.195264    mRNA
64165               41.982967           0.164837    mRNA
...                       ...                ...     ...
23308               38.836177           0.255514  lncRNA
5578                40.080033           0.241969  lncRNA
51197               40.633165           0.219493  lncRNA
21805               35.659156           0.225459  lncRNA
51734               44.510547           0.148362  lncRNA
```

[122600 rows x 50 columns]

6.6 Normalização dos dados treino

[122]:
```
#Transform categorical in binary class values
dicionario = {'mRNA':0,'lncRNA':1}
dadosTreino['label'] = dadosTreino['label'].map(dicionario)
```

[123]:
```
dadosTreino.columns
```

[123]:
```
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')
```

[124]:
```
#Removendo os campos nulos
dadosTreino.dropna(axis=1, inplace=True)
```

[125]:
```
#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)
```

[126]:
```
dadosTreino
```

[126]:
```
              A         C    D    E    F         G    H    I    K    L  ...  \
54832  0.265574  0.286885  0.0  0.0  0.0  0.232787  0.0  0.0  0.0  0.0  ...
29225  0.280387  0.230488  0.0  0.0  0.0  0.215683  0.0  0.0  0.0  0.0  ...
17358  0.281202  0.216487  0.0  0.0  0.0  0.228814  0.0  0.0  0.0  0.0  ...
31924  0.344602  0.179267  0.0  0.0  0.0  0.198730  0.0  0.0  0.0  0.0  ...
64165  0.257951  0.249117  0.0  0.0  0.0  0.265018  0.0  0.0  0.0  0.0  ...
...         ...       ...  ...  ...  ...       ...  ...  ...  ...  ...  ...
23308  0.288520  0.204431  0.0  0.0  0.0  0.197885  0.0  0.0  0.0  0.0  ...
5578   0.274664  0.226457  0.0  0.0  0.0  0.230381  0.0  0.0  0.0  0.0  ...
```

```
51197  0.266002  0.211416  0.0  0.0  0.0  0.234968  0.0  0.0  0.0  0.0  ...
21805  0.347059  0.179832  0.0  0.0  0.0  0.204202  0.0  0.0  0.0  0.0  ...
51734  0.320988  0.204938  0.0  0.0  0.0  0.214815  0.0  0.0  0.0  0.0  ...

       minimum_ORF_length  std_ORF_length  average_ORF_length  cv_ORF_length  \
54832            0.004545        0.016396            0.038832       0.045691
29225            0.018182        0.010353            0.045672       0.024529
17358            0.004545        0.254651            0.209656       0.131434
31924            0.009091        0.018381            0.081031       0.024546
64165            0.004545        0.004638            0.019890       0.025235
...                   ...             ...                 ...            ...
23308            0.004545        0.071291            0.064878       0.118909
5578             0.004545        0.028094            0.072176       0.042121
51197            0.004545        0.033106            0.058343       0.061403
21805            0.006818        0.013945            0.046851       0.032208
51734            0.015909        0.016106            0.060589       0.028764

       maximum_GC_content_ORF  minimum_GC_content_ORF  std_GC_content_ORF  \
54832                0.563636                0.196429            0.304750
29225                0.454545                0.413534            0.094012
17358                0.647727                0.314286            0.282014
31924                0.719192                0.491071            0.210586
64165                0.484848                0.392857            0.162445
...                       ...                     ...                 ...
23308                0.786241                0.130952            0.390109
5578                 0.675325                0.196429            0.375373
51197                0.534759                0.392857            0.177737
21805                0.588008                0.392857            0.242660
51734                0.588745                0.196429            0.341989

       average_GC_content_ORF  cv_GC_content_ORF  label
54832                0.440915           0.324290      0
29225                0.451195           0.097761      1
17358                0.579001           0.228526      1
31924                0.684401           0.144366      0
64165                0.466728           0.163301      0
...                       ...                ...    ...
23308                0.392339           0.466518      1
5578                 0.553832           0.318002      1
51197                0.464304           0.179605      1
21805                0.548167           0.207697      1
51734                0.521603           0.307622      1

[122600 rows x 50 columns]
```

[127]:
```
#Divide a base entre os previsores e classe
colunas = dadosTreino.columns.drop('label')
```

[128]:
```python
# Gera os previsores e classe (X e y)
X = dadosTreino[colunas].values
y = dadosTreino['label']
```

6.7 Dados de Teste

[129]:
```python
# 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])
```

[130]:
```python
dadosTeste
```

[130]:
```
                                                  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]
```

6.8 Redução da amostragem para 90%

[131]:
```
dadosTeste.shape
```

[131]: (58382, 51)

[132]:
```
dadosTeste = dadosTeste.sample(frac = 0.90)
```

[133]:
```
dadosTeste.shape
```

[133]: (52544, 51)

[134]:
```
#Remove column nameseq
dadosTeste.drop(columns='nameseq', inplace=True)
```

[135]:
```
dadosTeste.shape
```

[135]:
```
(52544, 50)
```

[136]:
```
#Transform categorical in binary class values
dicionario = {'mRNA':0,'lncRNA':1}
dadosTeste['label'] = dadosTeste['label'].map(dicionario)
```

[137]:
```
dadosTeste
```

[137]:
```
             A         C    D    E    F         G    H    I    K    L  ...  \
11042  0.245363  0.278230  0.0  0.0  0.0  0.252847  0.0  0.0  0.0  0.0  ...
25665  0.295652  0.207246  0.0  0.0  0.0  0.249275  0.0  0.0  0.0  0.0  ...
29178  0.229508  0.266393  0.0  0.0  0.0  0.322131  0.0  0.0  0.0  0.0  ...
1063   0.289431  0.191870  0.0  0.0  0.0  0.234146  0.0  0.0  0.0  0.0  ...
9861   0.258397  0.232188  0.0  0.0  0.0  0.256743  0.0  0.0  0.0  0.0  ...
...         ...       ...  ...  ...  ...       ...  ...  ...  ...  ...  ...
26322  0.336221  0.174337  0.0  0.0  0.0  0.215485  0.0  0.0  0.0  0.0  ...
24940  0.300250  0.230192  0.0  0.0  0.0  0.217681  0.0  0.0  0.0  0.0  ...
12319  0.324206  0.196057  0.0  0.0  0.0  0.193866  0.0  0.0  0.0  0.0  ...
26431  0.289806  0.228143  0.0  0.0  0.0  0.235147  0.0  0.0  0.0  0.0  ...
28672  0.221347  0.300087  0.0  0.0  0.0  0.265092  0.0  0.0  0.0  0.0  ...

       minimum_ORF_length  std_ORF_length  average_ORF_length  cv_ORF_length  \
11042                   6      676.021302          234.642857       2.881065
25665                   9       45.779637           76.615385       0.597525
29178                   6      168.403711          110.700000       1.521262
1063                    6       46.090780           43.200000       1.066916
9861                    6      320.008260           95.064516       3.366222
...                   ...             ...                 ...            ...
26322                   6       34.204324           48.827586       0.700512
24940                   6       44.732538           42.000000       1.065060
12319                   6       59.504354           61.235294       0.971733
26431                   6      616.037105          139.556962       4.414234
28672                  18      157.760071          200.400000       0.787226

       maximum_GC_content_ORF  minimum_GC_content_ORF  std_GC_content_ORF  \
11042               67.816092               22.222222           10.963217
25665               60.000000               22.222222            9.508462
29178               74.074074               16.666667           15.179602
1063                59.259259               32.142857           10.114475
9861                61.538462               16.666667            9.888295
...                       ...                     ...                 ...
26322               52.083333               10.000000           10.928127
24940               54.901961               11.111111           12.487487
12319               47.222222               26.666667            5.483812
26431               60.416667               14.285714            9.359862
```

```
28672              66.450216              43.902439              8.411655

       average_GC_content_ORF  cv_GC_content_ORF  label
11042               45.043311           0.243393      0
25665               42.626441           0.223065      1
29178               44.817392           0.338699      0
1063                39.640212           0.255157      1
9861                42.297585           0.233779      0
...                       ...                ...    ...
26322               34.236969           0.319191      1
24940               39.315916           0.317619      1
12319               36.785428           0.149076      1
26431               42.446158           0.220511      0
28672               56.051058           0.150071      1

[52544 rows x 50 columns]
```

[138]:

```
#Removendo os campos nulos
dadosTeste.dropna(axis=1, inplace=True)
```

6.9 Normalização dos dados Teste

[139]:

```
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)
```

[140]:

```
dadosTeste
```

[140]:

```
              A         C    D    E    F         G    H    I    K    L  ...  \
11042  0.245363  0.278230  0.0  0.0  0.0  0.252847  0.0  0.0  0.0  0.0  ...
25665  0.295652  0.207246  0.0  0.0  0.0  0.249275  0.0  0.0  0.0  0.0  ...
29178  0.229508  0.266393  0.0  0.0  0.0  0.322131  0.0  0.0  0.0  0.0  ...
1063   0.289431  0.191870  0.0  0.0  0.0  0.234146  0.0  0.0  0.0  0.0  ...
9861   0.258397  0.232188  0.0  0.0  0.0  0.256743  0.0  0.0  0.0  0.0  ...
...         ...       ...  ...  ...  ...       ...  ...  ...  ...  ...  ...
26322  0.336221  0.174337  0.0  0.0  0.0  0.215485  0.0  0.0  0.0  0.0  ...
24940  0.300250  0.230192  0.0  0.0  0.0  0.217681  0.0  0.0  0.0  0.0  ...
12319  0.324206  0.196057  0.0  0.0  0.0  0.193866  0.0  0.0  0.0  0.0  ...
26431  0.289806  0.228143  0.0  0.0  0.0  0.235147  0.0  0.0  0.0  0.0  ...
28672  0.221347  0.300087  0.0  0.0  0.0  0.265092  0.0  0.0  0.0  0.0  ...

       minimum_ORF_length  std_ORF_length  average_ORF_length  cv_ORF_length  \
11042            0.007059        0.024472            0.055294       0.049957
25665            0.004706        0.015875            0.042257       0.042406
29178            0.054118        0.040895            0.144314       0.031986
```

```
1063               0.007059      0.009953          0.039608      0.028365
9861               0.004706      0.027102          0.065882      0.046434
...                     ...           ...               ...           ...
26322              0.009412      0.096992          0.135529      0.080782
24940              0.030588      0.014759          0.061765      0.026973
12319              0.004706      0.081777          0.109916      0.083980
26431              0.004706      0.007865          0.030327      0.029275
28672              0.004706      0.023815          0.056676      0.047430

       maximum_GC_content_ORF  minimum_GC_content_ORF  std_GC_content_ORF  \
11042                0.503953                0.261905            0.275088
25665                0.615942                0.181319            0.288313
29178                0.626654                0.546167            0.139854
1063                 0.483278                0.261905            0.245664
9861                 0.554348                0.327381            0.192585
...                       ...                     ...                 ...
26322                0.739130                0.458333            0.243827
24940                0.554348                0.483516            0.126049
12319                0.698068                0.196429            0.451715
26431                0.638340                0.392857            0.280628
28672                0.665217                0.261905            0.310493

       average_GC_content_ORF  cv_GC_content_ORF  label
11042                0.412441           0.322295      0
25665                0.416664           0.334367      1
29178                0.608173           0.111120      0
1063                 0.419837           0.282752      1
9861                 0.478956           0.194299      0
...                       ...                ...    ...
26322                0.677745           0.173844      1
24940                0.534856           0.113880      1
12319                0.583051           0.374371      1
26431                0.507650           0.267122      0
28672                0.515475           0.291064      1

[52544 rows x 50 columns]
```

[141]:
```python
# Gera os previsores e classe (X e y)
X_teste = dadosTeste[colunas].values
y_teste = dadosTeste['label']
```

[142]:
```python
X_teste.shape
```

[142]: (52544, 49)

[143]:
```python
print(y_teste)
```

```
11042    0
```

```
25665    1
29178    0
1063     1
9861     0
        ..
26322    1
24940    1
12319    1
26431    0
28672    1
Name: label, Length: 52544, dtype: int64
```

[144]:
```
# 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')
```

[145]:
```
print(X.shape, y.shape, X_teste.shape, y_teste.shape)
```

```
(122600, 49) (122600,) (52544, 49) (52544,)
```

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

[151]:
```
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)
```

```
59.69
```

6.11 Aplica o modelo de predição com RandomForest e Wrapper

[147]:
```
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=24,step=1)
rfe = rfe.fit(X,y)
```

[148]:
```
#Armazena a nova dimensão do vetor de características
features = rfe.fit_transform(X,y)
```

[149]:
```
#Verifica a quantidade
print(features.shape)
```

```
(122600, 24)
```

6.12 Obtendo as melhores features

[152]:
```
temp = pandas.Series(rfe.support_,index = colunas)
wrapperApproach = temp[temp==True].index
print(wrapperApproach)
```

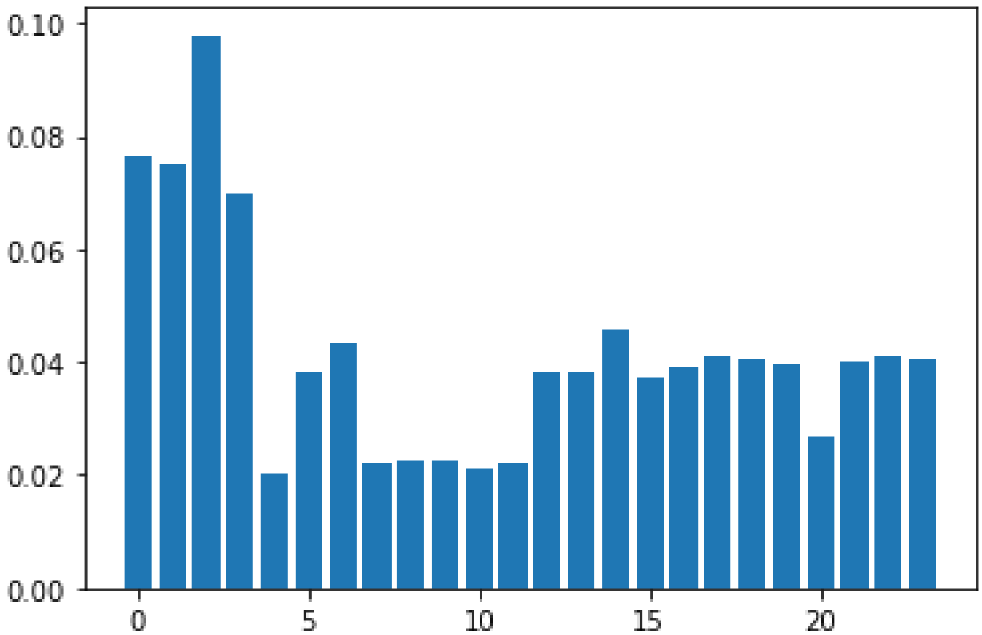
```
Index(['A', 'C', 'G', 'T', 'minimum', 'peak', 'none_levated_peak',
       'percentile15', 'percentile25', 'percentile50', 'percentile75',
       'semi_interquartile_range', 'coefficient_of_variation', 'skewness',
       'kurtosis', 'maximum_ORF_length', 'std_ORF_length',
       'average_ORF_length', 'cv_ORF_length', 'maximum_GC_content_ORF',
       'minimum_GC_content_ORF', 'std_GC_content_ORF',
       'average_GC_content_ORF', 'cv_GC_content_ORF'],
      dtype='object')
```

6.13 Feature Importance

[153]:
```
# 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_
```

```python
# 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
plt.bar([x for x in range(len(importance))], importance)
plt.show()
```

```
Feature A - score 0.07673
Feature C - score 0.07526
Feature G - score 0.09790
Feature T - score 0.06974
Feature minimum - score 0.02021
Feature peak - score 0.03800
Feature none_levated_peak - score 0.04343
Feature percentile15 - score 0.02217
Feature percentile25 - score 0.02244
Feature percentile50 - score 0.02249
Feature percentile75 - score 0.02106
Feature semi_interquartile_range - score 0.02218
Feature coefficient_of_variation - score 0.03840
Feature skewness - score 0.03800
Feature kurtosis - score 0.04586
Feature maximum_ORF_length - score 0.03742
Feature std_ORF_length - score 0.03898
Feature average_ORF_length - score 0.04094
Feature cv_ORF_length - score 0.04070
Feature maximum_GC_content_ORF - score 0.03952
Feature minimum_GC_content_ORF - score 0.02677
Feature std_GC_content_ORF - score 0.04024
Feature average_GC_content_ORF - score 0.04110
Feature cv_GC_content_ORF - score 0.04044
```



[154]:
```
#Predicao sem validação cruzada
y_pred = rfe.predict(X_teste)
acuracidade = round(accuracy_score(y_teste,y_pred)*100,2)
print(acuracidade)
```

60.07

6.14 Validação cruzada no conjunto reduzido

[ ]:
```
kfold = KFold(n_splits=10, shuffle=True, random_state=123)
resultado = cross_val_score(rfe, X, y, cv=kfold, scoring='accuracy')
```

[ ]:
```
print('O score cross-validado do Random Forest é:', resultado.mean())
```

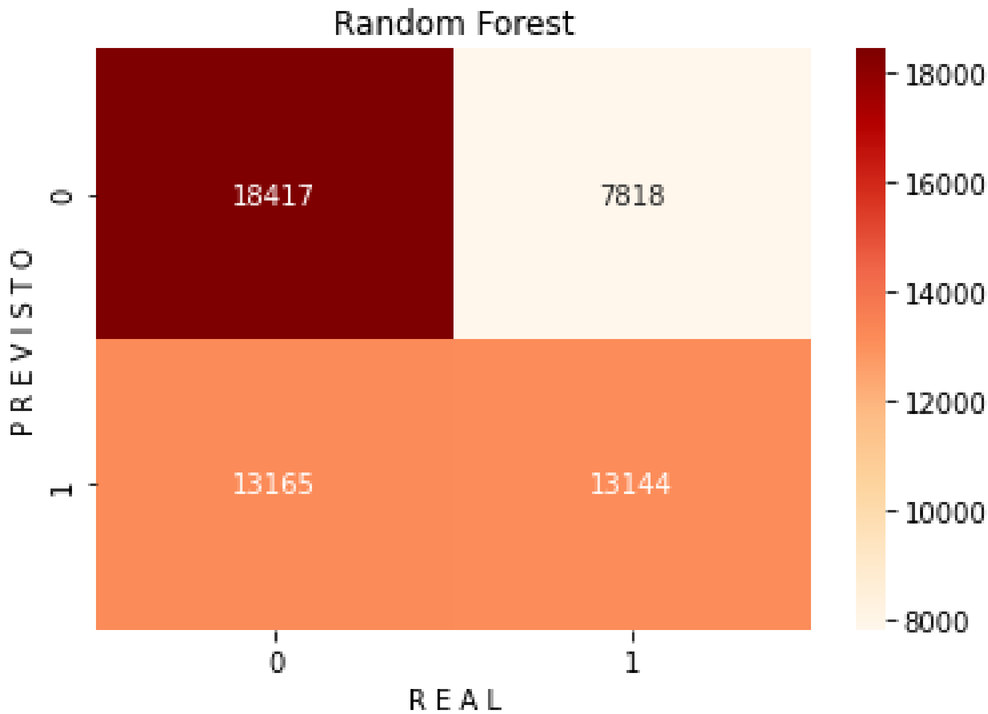
[ ]:
```
resultado
```

[156]:
```
rf_pred = rfe.predict(X_teste)
```

6.15 Calculando as métricas

[157]:
```
#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()
```



[171]:
```
#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)}')
```

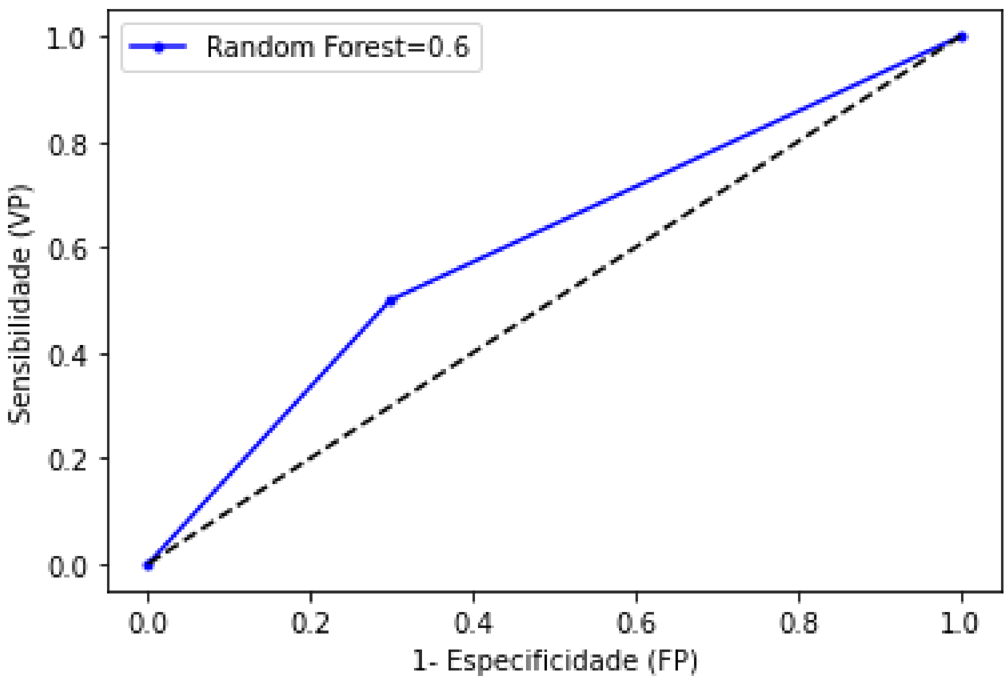
```
Acurácia:0.6
Especificidade:0.7
Precisão:0.63
```

```
Recall ou Sensibilidade:0.5
F1-Score:0.56
Kappa:0.2
Curva ROC:0.6
```

6.16 Curva ROC

[172]:
```
import matplotlib.pyplot  as pyplot
```

[173]:
```
rfp_rf, rvp_rf,lim2 = roc_curve(y_teste,rf_pred)
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()
```



6.17 Aplica o modelo de predição com Regressão Logística e Wrapper

[174]:
```
clf_rl = LogisticRegression(max_iter=2000)
rfe_rl = RFE(clf_rl,n_features_to_select=24,step=1)
fit_rl = rfe_rl.fit(X,y)
```

[175]:
```
#Armazena a nova dimensão do vetor de características
features_rl = fit_rl.fit_transform(X,y)
```

[176]:
```
#Verifica a quantidade
print(features_rl.shape)
```

```
(122600, 24)
```

6.18 Exibindo as melhores features

[177]:
```
temp_rl = pandas.Series(fit_rl.support_,index = colunas)
wrapperApproach_rl = temp_rl[temp_rl==True].index
print(wrapperApproach_rl)
```

```
Index(['A', 'C', 'G', 'T', 'median', 'peak', 'none_levated_peak',
       'percentile15', 'percentile25', 'percentile50', 'percentile75',
       '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'],
      dtype='object')
```

[178]:
```
#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)
```

```
50.03
```

6.19 Validação cruzada no conjunto reduzido

[231]:
```
kfold = KFold(n_splits=10, shuffle=True, random_state=123)
resultado_rl = cross_val_score(fit_rl, X, y, cv=kfold, scoring='accuracy')
```

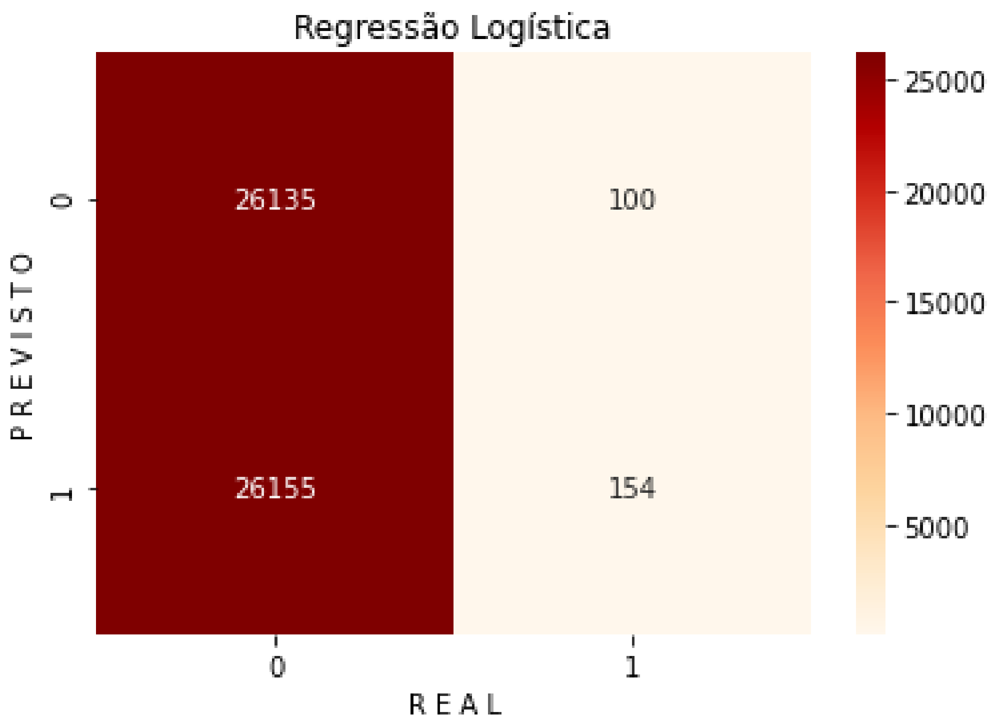
[ ]:
```
print('O score cross-validado do Regressão Logística é:', resultado_rl.mean())
```

[ ]:
```
resultado_rl
```

[179]:
```
rl_pred = fit_rl.predict(X_teste)
```

6.20 Calculando as métricas

[180]:
```
#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()
```



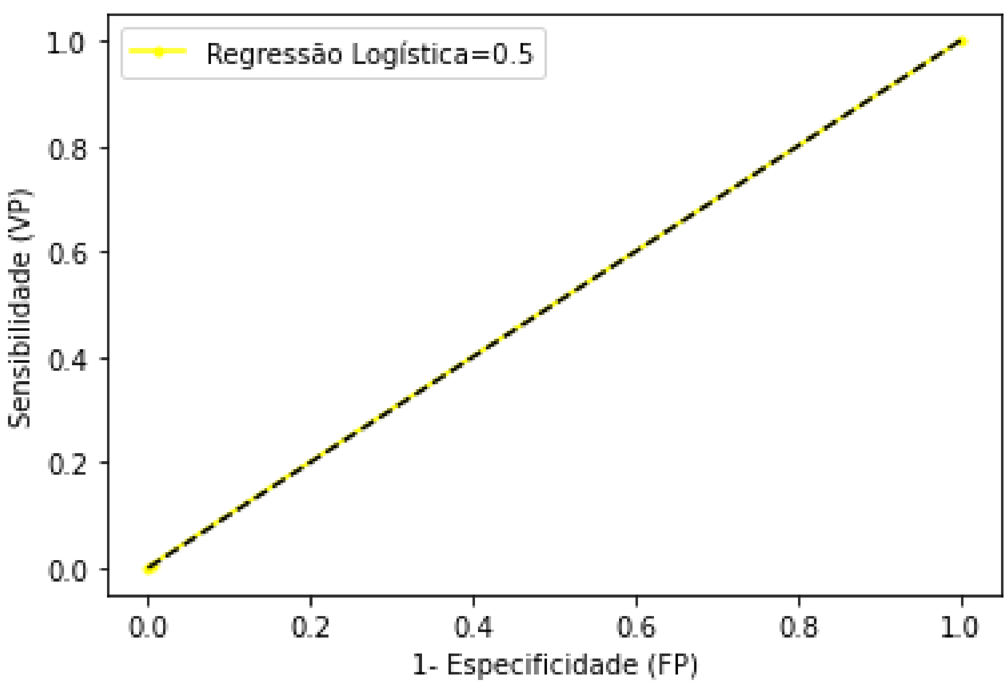
[181]:
```
#Acurácia, Sensibilidade positiva (VP/(VP+FN), Especificidade, Precisão, Recall,␣
↪F1-Score
acuracia_rl = accuracy_score(y_teste,rl_pred)
especificidade_rl = specificity_score(y_teste,rl_pred)
precisao_rl = precision_score(y_teste,rl_pred)
recall_rl = recall_score(y_teste,rl_pred)
f1Score_rl = 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:0.61
Recall ou Sensibilidade:0.01
F1-Score:0.01
Kappa:0.0
Curva ROC:0.5
```

6.21 Curva ROC

[182]:
```
rfp_rl, rvp_rl,lim1 = roc_curve(y_teste,rl_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()
```



6.22 Aplica o modelo de predição com SVM e Wrapper

[ ]:
```
from sklearn.svm import SVC
svc = SVC(C=1, kernel='linear')
rfe = RFE(estimator=svc, n_features_to_select=10, step=0.1)
fit_svm = rfe.fit(x_train,y_train)
```

[ ]:
```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
param_grid = {'C':[1e3,5e3,1e4,5e4,1e5],'gama':[0.0001,0.0005,0.001,0.005,0.1]}
svc = SVC()
clf_svm = GridSearchCV(svc, parameters)
#clf_svm = GridSearchCV(SVC(kernel='rbf'),param_grid)
fit_svm = clf_svm.fit(X,y)
#print('Best estimator found by GridSearch')
#print(clf_svm.best_estimator_)
#clf_svm = SVC(gamma='auto')
#rfe_svm = RFE(clf_svm, n_features_to_select=10, step=1)
#fit_svm = rfe_svm.fit(X,y)
```

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

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

6.23 Obtendo as melhores feature do modelo

[ ]:
```
temp_svm = pandas.Series(fit_svm.support_,index = colunas)
wrapperApproach_svm = temp_svm[temp_svm==True].index
print(wrapperApproach_svm)
```

[ ]:
```
#Predicao sem validação cruzada
y_pred_svm = fit_svm.predict(X_teste)
acuracidade_svm = round(accuracy_score(y_teste,y_pred_svm)*100,2)
print(acuracidade_svm)
```

6.24 Calculando as métricas

[ ]:
```
#Matriz de confusão
sns.heatmap(confusion_matrix(y_teste, svm_pred), cmap='OrRd', annot=True, fmt='2.
 ↪0f')
plt.title('SVM')
plt.ylabel('P R E V I S T O')
```

```
plt.xlabel('R E A L')
plt.show()
```

[ ]:
```
#Acurácia, Sensibilidade positiva (VP/(VP+FN), Especificidade, Precisão, Recall,␣
↪F1-Score
acuracia_svm = accuracy_score(y_teste,svm_pred)
especificidade_svm = specificity_score(y_teste,svm_pred)
precisao_svm = precision_score(y_teste,svm_pred)
recall_svm = recall_score(y_teste,svm_pred)
f1Score_svm = f1_score(y_teste,svm_pred)
curva_roc_escore_svm = roc_auc_score(y_teste,svm_pred)
kappa_svm = cohen_kappa_score(y_teste,svm_pred)
print(f'Acurácia:{round(acuracia_rl,2)}')
print(f'Especificidade:{round(especificidade_svm,2)}')
print(f'Precisão:{round(precisao_rl,2)}')
print(f'Recall ou Sensibilidade:{round(recall_svm,2)}')
print(f'F1-Score:{round(f1Score_svm,2)}')
print(f'Kappa:{round(kappa_svm,2)}')
print(f'Curva ROC:{round(curva_roc_escore_svm,2)}')
```

6.25 Curva ROC

[ ]:
```
rfp_rl, rvp_rl,lim1 = roc_curve(y_teste,rl_pred)
rfp_rf, rvp_rf,lim2 = roc_curve(y_teste,rf_pred)
rfp_svm, rvp_svm,lim3 = roc_curve(y_teste,svm_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(rfp_svm, rvp_svm, marker='.',␣
↪label='SVM='+str(round(curva_roc_escore_svm,2)),color='red')
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()
```

6.26 Aplica o modelo de predição com RandomForest e Filtro

[192]:
```
# 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=24)
selector.fit(X, y)
```

[192]: SelectKBest(k=24, score_func=<function mutual_info_classif at 0x7f98f91a7dc0>)

[193]:
```
# to remove the rest of the features:
X_train_filtro = selector.transform(X)
X_teste_filtro = selector.transform(X_teste)
```

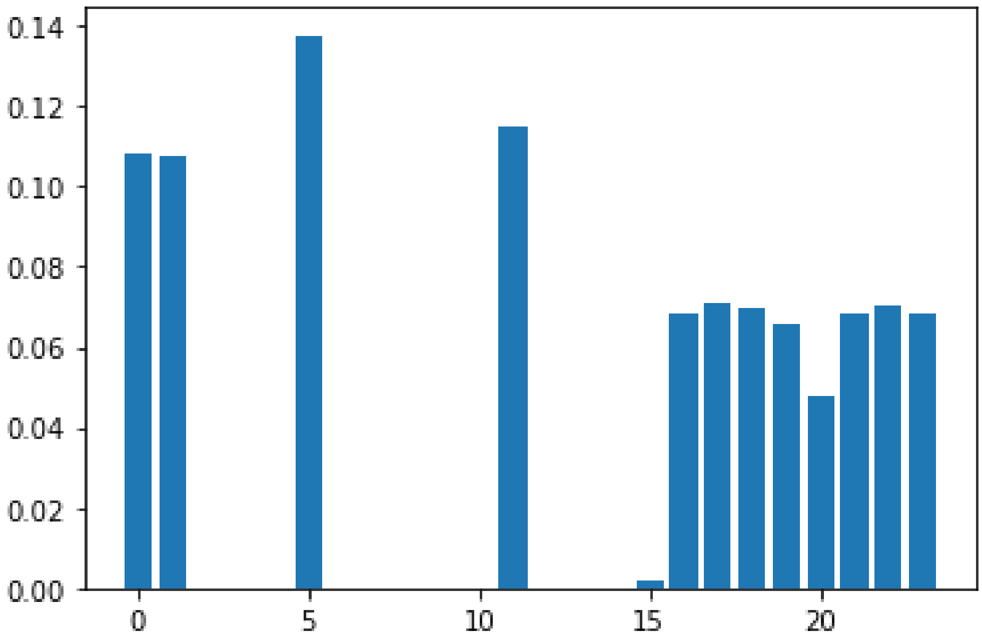
[194]:
```
#Executando o modelo
clf_rf_filtro = RandomForestClassifier(random_state=123)
clr_rf_filtro = clf_rf_filtro.fit(X_train_filtro,y)
```

[195]:
```
#Predição
rf_pred_filtro = clr_rf_filtro.predict(X_teste_filtro)
```

6.27 Feature importance

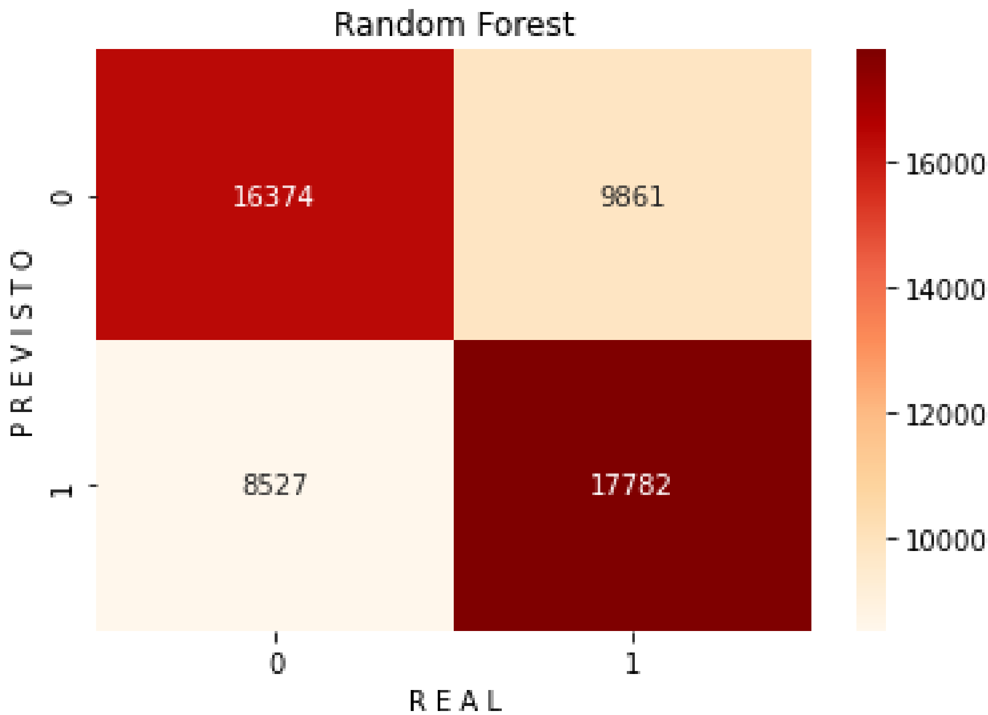
[196]:
```
colNames = dadosTreino.columns.tolist()
```

[197]:
```
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
plt.bar([x for x in range(len(importance))], importance)
plt.show()
```



6.28 Calculando as métricas

[198]:
```
#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()
```



[199]:
```
#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:0.65
Especificidade:0.62
Precisão:0.64
Recall ou Sensibilidade:0.68
F1-Score:0.66
Kappa:0.3
Curva ROC:0.65
```

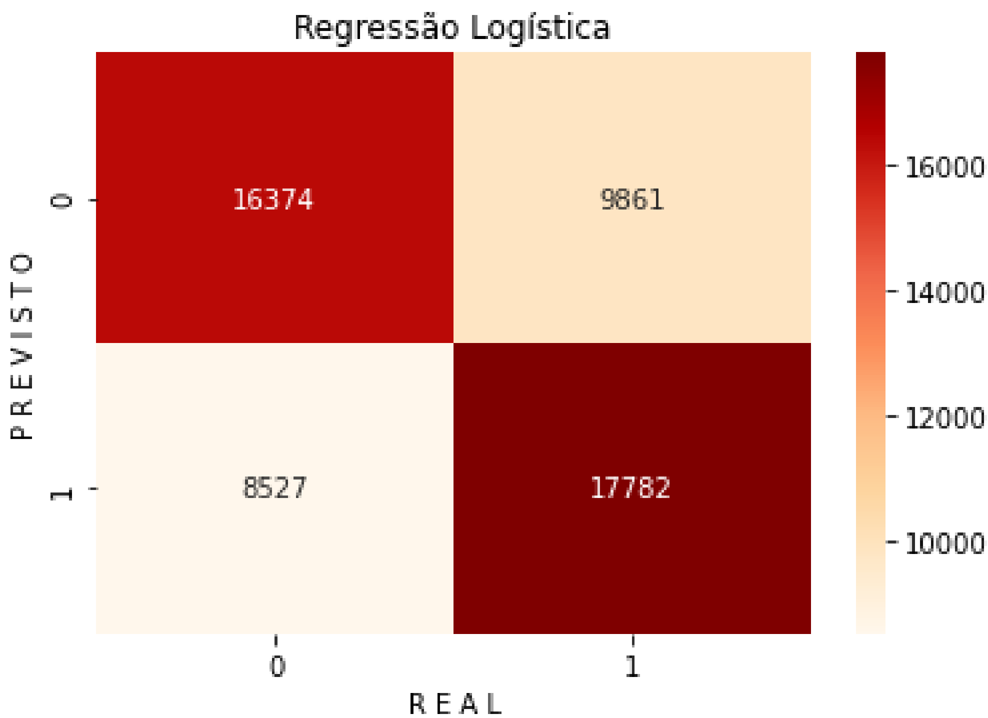
6.29 Aplica o modelo de predição com Regressão Logística e Filter

[200]:
```
#Executando o modelo
clf_rl_filtro = LogisticRegression(max_iter=2000)
clr_rl_filtro = clf_rf_filtro.fit(X_train_filtro,y)
```

[201]:
```
#Predição
rl_pred_filtro = clr_rl_filtro.predict(X_teste_filtro)
```

6.30 Calculando as métricas

[202]:
```
#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()
```



[203]:
```
#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:0.65
Especificidade:0.62
Precisão:0.64
Recall ou Sensibilidade:0.68
F1-Score:0.66
Kappa:0.3
Curva ROC:0.65
```

6.31 Aplica o modelo de predição SVM com Hiperparâmetros e Filter

[ ]:
```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
svc = SVC()
clf_svm_filtro = GridSearchCV(svc, parameters)
clf_svm_filtro = clf_svm_filtro.fit(X_train_filtro,y)
```

[ ]:
```
#Predição
svm_pred_filtro = clf_svm_filtro.predict(X_teste_filtro)
```

6.32 Calculando as métricas

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

[ ]:
```
#Acurácia, Sensibilidade positiva (VP/(VP+FN), Especificidade, Precisão, Recall,␣
 ↪F1-Score
acuracia_svm_f = accuracy_score(y_teste,svm_pred_filtro)
```

```
especificidade_svm_f = specificity_score(y_teste,svm_pred_filtro)
precisao_svm_f = precision_score(y_teste,svm_pred_filtro)
recall_svm_f = recall_score(y_teste,svm_pred_filtro)
f1Score_svm_f = f1_score(y_teste,svm_pred_filtro)
curva_roc_escore_svm_f = roc_auc_score(y_teste,svm_pred_filtro)
kappa_svm_f = cohen_kappa_score(y_teste,svm_pred_filtro)
print(f'Acurácia:{round(acuracia_svm_f,2)}')
print(f'Especificidade:{round(especificidade_svm_f,2)}')
print(f'Precisão:{round(precisao_svm_f,2)}')
print(f'Recall ou Sensibilidade:{round(recall_svm_f,2)}')
print(f'F1-Score:{round(f1Score_svm_f,2)}')
print(f'Kappa:{round(kappa_svm_f,2)}')
print(f'Curva ROC:{round(curva_roc_escore_svm_f,2)}')
```

6.33 Curva ROC

[ ]:
```
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)
rfp_svm_f,rvp_svm_f,lim6 = roc_curve(y_teste,svm_pred_filtro)
pyplot.plot(rfp_rl, rvp_rl, marker='.', label='Regressão␣
↪Logística='+str(round(curva_roc_escore_rl_f,2)),color='yellow')
pyplot.plot(rfp_rf, rvp_rf, 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()
```

7 Autoencoders

[205]:
```
# Importanfo as bibliotecas
from sklearn.datasets import make_classification
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.utils import plot_model
from matplotlib import pyplot
```

[206]:
```
#Verificando X e Y
print(X.shape, y.shape, X_teste.shape, y_teste.shape)
```

```
(122600, 49) (122600,) (52544, 49) (52544,)
```

[207]:
```
#Pegando os números de input
n_inputs = X.shape[1]
#definindo o encoder
visible = Input(shape=(n_inputs,))
```

[208]:
```
#Encoder nível 1. Definindo a primeira camada oculta
e = Dense(n_inputs*2)(visible)
#Usando a nomalização em lote para garantir que o modelo aprenda bem
e = BatchNormalization()(e)
#Definindo a função de ativação Relu
e = LeakyReLU()(e)
```

[209]:
```
# Encoder nível 2. Definindo a segunda camada oculta
e = Dense(n_inputs)(visible)
#Usando a nomalização em lote para garantir que o modelo aprenda bem
e = BatchNormalization()(e)
#Definindo a função de ativação Relu
e = LeakyReLU()(e)
```

[210]:
```
#Camada de redução. aqui que acontece a redução
#n_bottleneck = round(float(n_inputs) / 2.0)
n_bottleneck = 24
bottleneck = Dense(n_bottleneck)(e)
```

[211]:
```
#Definindo o decoder, level 1
d = Dense(n_inputs)(bottleneck)
d = BatchNormalization()(d)
d = LeakyReLU()(d)
```

[212]:
```
#Definindo o decoder nível 2
d = Dense(n_inputs*2)(d)
d = BatchNormalization()(d)
d = LeakyReLU()(d)
```

[213]:
```
#Camada de saída usando a função de ativação
#É a função mais básica porque não altera a saída de um neurônio
output = Dense(n_inputs, activation='linear')(d)
```

[214]:
```
#definindo o modelo de autoencoder model
model = Model(inputs=visible, outputs=output)
```

[215]:
```
#Compilando o modelo autoencoder
```

```
#adam =  função com base no método de descida gradiente estocástico. Tende a␣
↪convergir rapidamente.
#binary_crossentropy  = um função utilizada para problemas de classificação␣
↪binária (0 ou 1)
# mse = calcula a média dos quadrados dos erros entre rótulos e previsões
#model.compile(optimizer='adam', loss='binary_crossentropy')
model.compile(optimizer='adam', loss='mse')
```

[216]:
```
#Ajustar o modelo autoencoder para reconstruir a entrada
history = model.fit(X,X, epochs=200, batch_size=16, verbose=2,␣
↪validation_data=(X_teste,X_teste))
```

```
Epoch 1/200
7663/7663 - 22s - loss: 6.7119 - val_loss: 164.7532
Epoch 2/200
7663/7663 - 18s - loss: 1.9550 - val_loss: 94.8714
Epoch 3/200
7663/7663 - 17s - loss: 1.2577 - val_loss: 76.7719
Epoch 4/200
7663/7663 - 20s - loss: 1.2238 - val_loss: 63.1273
Epoch 5/200
7663/7663 - 21s - loss: 1.1335 - val_loss: 44.5124
Epoch 6/200
7663/7663 - 26s - loss: 1.1085 - val_loss: 40.9101
Epoch 7/200
7663/7663 - 25s - loss: 1.1711 - val_loss: 26.3052
Epoch 8/200
7663/7663 - 20s - loss: 1.1336 - val_loss: 21.7634
Epoch 9/200
7663/7663 - 17s - loss: 0.9978 - val_loss: 28.0016
Epoch 10/200
7663/7663 - 17s - loss: 0.9096 - val_loss: 39.9975
Epoch 11/200
7663/7663 - 17s - loss: 0.8601 - val_loss: 13.1212
Epoch 12/200
7663/7663 - 17s - loss: 0.7742 - val_loss: 35.3156
Epoch 13/200
7663/7663 - 17s - loss: 0.7087 - val_loss: 16.5946
Epoch 14/200
7663/7663 - 17s - loss: 0.7228 - val_loss: 26.9248
Epoch 15/200
7663/7663 - 17s - loss: 0.6555 - val_loss: 14.0932
Epoch 16/200
7663/7663 - 18s - loss: 0.6250 - val_loss: 16.2523
Epoch 17/200
7663/7663 - 17s - loss: 0.5926 - val_loss: 53.9431
Epoch 18/200
7663/7663 - 17s - loss: 0.5571 - val_loss: 34.9339
```

```
Epoch 19/200
7663/7663 - 17s - loss: 0.5101 - val_loss: 19.3192
Epoch 20/200
7663/7663 - 17s - loss: 0.5638 - val_loss: 15.4115
Epoch 21/200
7663/7663 - 17s - loss: 0.4997 - val_loss: 28.4237
Epoch 22/200
7663/7663 - 17s - loss: 0.4743 - val_loss: 41.4573
Epoch 23/200
7663/7663 - 18s - loss: 0.5020 - val_loss: 19.7286
Epoch 24/200
7663/7663 - 17s - loss: 0.4952 - val_loss: 21.9825
Epoch 25/200
7663/7663 - 17s - loss: 0.4692 - val_loss: 26.1841
Epoch 26/200
7663/7663 - 18s - loss: 0.4409 - val_loss: 9.6909
Epoch 27/200
7663/7663 - 17s - loss: 0.4409 - val_loss: 6.7735
Epoch 28/200
7663/7663 - 17s - loss: 0.4288 - val_loss: 15.7345
Epoch 29/200
7663/7663 - 17s - loss: 0.3832 - val_loss: 22.1298
Epoch 30/200
7663/7663 - 18s - loss: 0.4229 - val_loss: 21.3504
Epoch 31/200
7663/7663 - 17s - loss: 0.4116 - val_loss: 5.6749
Epoch 32/200
7663/7663 - 17s - loss: 0.4400 - val_loss: 14.2268
Epoch 33/200
7663/7663 - 17s - loss: 0.3931 - val_loss: 8.0682
Epoch 34/200
7663/7663 - 17s - loss: 0.3729 - val_loss: 11.2109
Epoch 35/200
7663/7663 - 17s - loss: 0.3631 - val_loss: 6.2583
Epoch 36/200
7663/7663 - 17s - loss: 0.3424 - val_loss: 7.1607
Epoch 37/200
7663/7663 - 17s - loss: 0.3552 - val_loss: 19.0599
Epoch 38/200
7663/7663 - 17s - loss: 0.3604 - val_loss: 18.9623
Epoch 39/200
7663/7663 - 18s - loss: 0.3656 - val_loss: 9.6864
Epoch 40/200
7663/7663 - 17s - loss: 0.3575 - val_loss: 30.5079
Epoch 41/200
7663/7663 - 17s - loss: 0.3290 - val_loss: 10.3574
Epoch 42/200
7663/7663 - 18s - loss: 0.3831 - val_loss: 13.3867
```

```
Epoch 43/200
7663/7663 - 18s - loss: 0.3247 - val_loss: 7.1255
Epoch 44/200
7663/7663 - 18s - loss: 0.3434 - val_loss: 10.7153
Epoch 45/200
7663/7663 - 19s - loss: 0.3136 - val_loss: 7.0252
Epoch 46/200
7663/7663 - 18s - loss: 0.3201 - val_loss: 9.2920
Epoch 47/200
7663/7663 - 17s - loss: 0.3290 - val_loss: 4.3554
Epoch 48/200
7663/7663 - 17s - loss: 0.3119 - val_loss: 11.4076
Epoch 49/200
7663/7663 - 17s - loss: 0.3021 - val_loss: 12.2028
Epoch 50/200
7663/7663 - 17s - loss: 0.3138 - val_loss: 6.8211
Epoch 51/200
7663/7663 - 17s - loss: 0.2982 - val_loss: 3.4457
Epoch 52/200
7663/7663 - 17s - loss: 0.3070 - val_loss: 8.8568
Epoch 53/200
7663/7663 - 17s - loss: 0.3083 - val_loss: 5.3641
Epoch 54/200
7663/7663 - 17s - loss: 0.3091 - val_loss: 2.6837
Epoch 55/200
7663/7663 - 17s - loss: 0.2874 - val_loss: 3.9509
Epoch 56/200
7663/7663 - 19s - loss: 0.2898 - val_loss: 4.1960
Epoch 57/200
7663/7663 - 18s - loss: 0.3015 - val_loss: 3.5165
Epoch 58/200
7663/7663 - 20s - loss: 0.2787 - val_loss: 3.0885
Epoch 59/200
7663/7663 - 18s - loss: 0.3055 - val_loss: 4.1823
Epoch 60/200
7663/7663 - 17s - loss: 0.2967 - val_loss: 4.6880
Epoch 61/200
7663/7663 - 17s - loss: 0.2872 - val_loss: 2.4323
Epoch 62/200
7663/7663 - 18s - loss: 0.3141 - val_loss: 4.7987
Epoch 63/200
7663/7663 - 16s - loss: 0.2691 - val_loss: 2.2742
Epoch 64/200
7663/7663 - 16s - loss: 0.2899 - val_loss: 2.1423
Epoch 65/200
7663/7663 - 16s - loss: 0.2684 - val_loss: 2.2729
Epoch 66/200
7663/7663 - 19s - loss: 0.3022 - val_loss: 2.3601
```

```
Epoch 67/200
7663/7663 - 18s - loss: 0.2797 - val_loss: 6.1292
Epoch 68/200
7663/7663 - 17s - loss: 0.2848 - val_loss: 9.2903
Epoch 69/200
7663/7663 - 20s - loss: 0.2715 - val_loss: 6.8184
Epoch 70/200
7663/7663 - 21s - loss: 0.2752 - val_loss: 2.8667
Epoch 71/200
7663/7663 - 23s - loss: 0.2577 - val_loss: 4.3590
Epoch 72/200
7663/7663 - 18s - loss: 0.2540 - val_loss: 3.8142
Epoch 73/200
7663/7663 - 22s - loss: 0.2742 - val_loss: 3.2769
Epoch 74/200
7663/7663 - 21s - loss: 0.2637 - val_loss: 3.7487
Epoch 75/200
7663/7663 - 18s - loss: 0.2825 - val_loss: 3.7312
Epoch 76/200
7663/7663 - 18s - loss: 0.2541 - val_loss: 4.1975
Epoch 77/200
7663/7663 - 17s - loss: 0.2677 - val_loss: 2.8445
Epoch 78/200
7663/7663 - 18s - loss: 0.2662 - val_loss: 2.9770
Epoch 79/200
7663/7663 - 18s - loss: 0.2762 - val_loss: 4.0044
Epoch 80/200
7663/7663 - 25s - loss: 0.2510 - val_loss: 3.7606
Epoch 81/200
7663/7663 - 18s - loss: 0.2495 - val_loss: 2.7813
Epoch 82/200
7663/7663 - 17s - loss: 0.2736 - val_loss: 2.1631
Epoch 83/200
7663/7663 - 18s - loss: 0.2496 - val_loss: 2.3194
Epoch 84/200
7663/7663 - 20s - loss: 0.2367 - val_loss: 7.6189
Epoch 85/200
7663/7663 - 17s - loss: 0.2697 - val_loss: 2.8145
Epoch 86/200
7663/7663 - 18s - loss: 0.2806 - val_loss: 3.6866
Epoch 87/200
7663/7663 - 18s - loss: 0.2501 - val_loss: 3.0629
Epoch 88/200
7663/7663 - 22s - loss: 0.2490 - val_loss: 2.8909
Epoch 89/200
7663/7663 - 19s - loss: 0.2458 - val_loss: 3.1759
Epoch 90/200
7663/7663 - 17s - loss: 0.2377 - val_loss: 2.6914
```

```
Epoch 91/200
7663/7663 - 17s - loss: 0.2421 - val_loss: 2.7804
Epoch 92/200
7663/7663 - 20s - loss: 0.2416 - val_loss: 2.8794
Epoch 93/200
7663/7663 - 18s - loss: 0.2487 - val_loss: 2.5056
Epoch 94/200
7663/7663 - 16s - loss: 0.2534 - val_loss: 2.5993
Epoch 95/200
7663/7663 - 16s - loss: 0.2418 - val_loss: 2.2708
Epoch 96/200
7663/7663 - 16s - loss: 0.2456 - val_loss: 4.2732
Epoch 97/200
7663/7663 - 17s - loss: 0.2461 - val_loss: 2.9463
Epoch 98/200
7663/7663 - 17s - loss: 0.2403 - val_loss: 5.3785
Epoch 99/200
7663/7663 - 16s - loss: 0.2426 - val_loss: 2.7584
Epoch 100/200
7663/7663 - 16s - loss: 0.2630 - val_loss: 2.6190
Epoch 101/200
7663/7663 - 16s - loss: 0.2086 - val_loss: 3.3929
Epoch 102/200
7663/7663 - 16s - loss: 0.2258 - val_loss: 3.3844
Epoch 103/200
7663/7663 - 16s - loss: 0.2288 - val_loss: 4.6425
Epoch 104/200
7663/7663 - 16s - loss: 0.2289 - val_loss: 2.9214
Epoch 105/200
7663/7663 - 16s - loss: 0.2414 - val_loss: 4.0942
Epoch 106/200
7663/7663 - 17s - loss: 0.2399 - val_loss: 3.4044
Epoch 107/200
7663/7663 - 16s - loss: 0.2454 - val_loss: 3.4314
Epoch 108/200
7663/7663 - 16s - loss: 0.2289 - val_loss: 3.0127
Epoch 109/200
7663/7663 - 16s - loss: 0.2368 - val_loss: 3.2507
Epoch 110/200
7663/7663 - 16s - loss: 0.2266 - val_loss: 2.7600
Epoch 111/200
7663/7663 - 20s - loss: 0.2618 - val_loss: 3.8558
Epoch 112/200
7663/7663 - 18s - loss: 0.2222 - val_loss: 3.0070
Epoch 113/200
7663/7663 - 16s - loss: 0.2385 - val_loss: 3.6469
Epoch 114/200
7663/7663 - 19s - loss: 0.2163 - val_loss: 3.2087
```

```
Epoch 115/200
7663/7663 - 17s - loss: 0.2273 - val_loss: 4.6774
Epoch 116/200
7663/7663 - 17s - loss: 0.2318 - val_loss: 3.2873
Epoch 117/200
7663/7663 - 17s - loss: 0.2471 - val_loss: 3.9040
Epoch 118/200
7663/7663 - 17s - loss: 0.2300 - val_loss: 3.7317
Epoch 119/200
7663/7663 - 16s - loss: 0.2402 - val_loss: 3.2305
Epoch 120/200
7663/7663 - 17s - loss: 0.2150 - val_loss: 3.6613
Epoch 121/200
7663/7663 - 16s - loss: 0.2388 - val_loss: 3.7468
Epoch 122/200
7663/7663 - 17s - loss: 0.2555 - val_loss: 3.0812
Epoch 123/200
7663/7663 - 17s - loss: 0.2271 - val_loss: 2.8088
Epoch 124/200
7663/7663 - 17s - loss: 0.2233 - val_loss: 3.4352
Epoch 125/200
7663/7663 - 19s - loss: 0.2316 - val_loss: 3.4817
Epoch 126/200
7663/7663 - 17s - loss: 0.2197 - val_loss: 3.1180
Epoch 127/200
7663/7663 - 17s - loss: 0.2089 - val_loss: 3.7720
Epoch 128/200
7663/7663 - 16s - loss: 0.2281 - val_loss: 3.6769
Epoch 129/200
7663/7663 - 17s - loss: 0.2289 - val_loss: 2.9042
Epoch 130/200
7663/7663 - 16s - loss: 0.2387 - val_loss: 2.9636
Epoch 131/200
7663/7663 - 16s - loss: 0.2310 - val_loss: 3.2941
Epoch 132/200
7663/7663 - 17s - loss: 0.2283 - val_loss: 3.9180
Epoch 133/200
7663/7663 - 19s - loss: 0.2162 - val_loss: 3.7304
Epoch 134/200
7663/7663 - 17s - loss: 0.2410 - val_loss: 3.5848
Epoch 135/200
7663/7663 - 17s - loss: 0.2381 - val_loss: 3.0619
Epoch 136/200
7663/7663 - 16s - loss: 0.2165 - val_loss: 3.3166
Epoch 137/200
7663/7663 - 16s - loss: 0.2176 - val_loss: 3.1800
Epoch 138/200
7663/7663 - 16s - loss: 0.2270 - val_loss: 4.4058
```

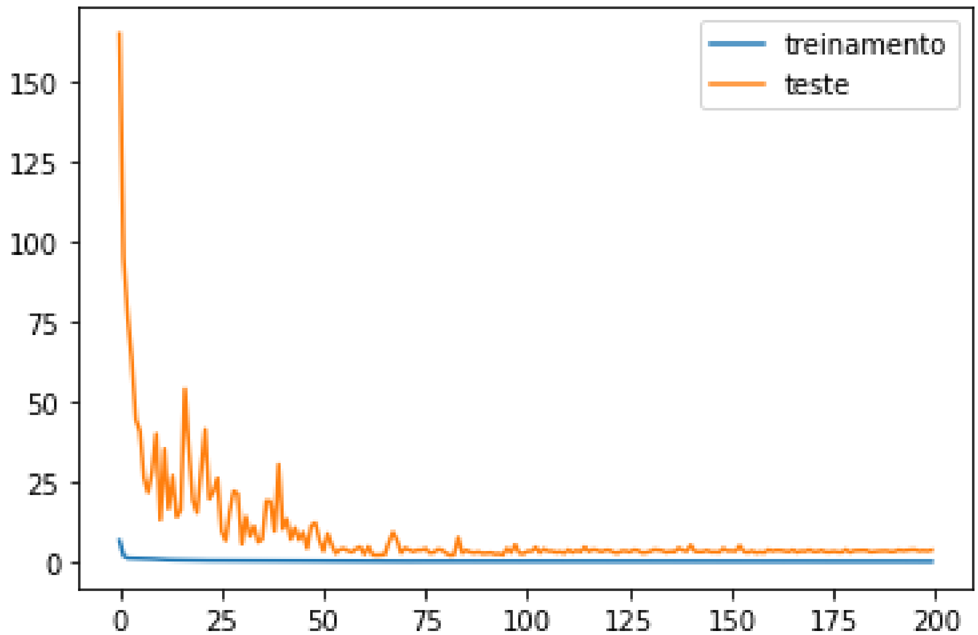
```
Epoch 139/200
7663/7663 - 17s - loss: 0.2210 - val_loss: 3.7110
Epoch 140/200
7663/7663 - 19s - loss: 0.2272 - val_loss: 3.3409
Epoch 141/200
7663/7663 - 17s - loss: 0.2209 - val_loss: 5.2368
Epoch 142/200
7663/7663 - 16s - loss: 0.2453 - val_loss: 3.3291
Epoch 143/200
7663/7663 - 20s - loss: 0.2369 - val_loss: 3.2128
Epoch 144/200
7663/7663 - 21s - loss: 0.2117 - val_loss: 3.2490
Epoch 145/200
7663/7663 - 22s - loss: 0.2043 - val_loss: 3.6335
Epoch 146/200
7663/7663 - 21s - loss: 0.2232 - val_loss: 3.1279
Epoch 147/200
7663/7663 - 21s - loss: 0.2086 - val_loss: 3.1918
Epoch 148/200
7663/7663 - 22s - loss: 0.2242 - val_loss: 3.0491
Epoch 149/200
7663/7663 - 23s - loss: 0.2116 - val_loss: 4.2630
Epoch 150/200
7663/7663 - 21s - loss: 0.2414 - val_loss: 3.4709
Epoch 151/200
7663/7663 - 21s - loss: 0.1882 - val_loss: 3.5411
Epoch 152/200
7663/7663 - 21s - loss: 0.2235 - val_loss: 3.3644
Epoch 153/200
7663/7663 - 21s - loss: 0.2215 - val_loss: 5.0962
Epoch 154/200
7663/7663 - 20s - loss: 0.2077 - val_loss: 3.3082
Epoch 155/200
7663/7663 - 21s - loss: 0.2132 - val_loss: 2.9964
Epoch 156/200
7663/7663 - 26s - loss: 0.2212 - val_loss: 3.5159
Epoch 157/200
7663/7663 - 26s - loss: 0.2194 - val_loss: 2.8936
Epoch 158/200
7663/7663 - 30s - loss: 0.2142 - val_loss: 3.1638
Epoch 159/200
7663/7663 - 34s - loss: 0.2194 - val_loss: 2.9317
Epoch 160/200
7663/7663 - 42s - loss: 0.2146 - val_loss: 3.8762
Epoch 161/200
7663/7663 - 26s - loss: 0.2251 - val_loss: 3.4432
Epoch 162/200
7663/7663 - 29s - loss: 0.1942 - val_loss: 3.7062
```

```
Epoch 163/200
7663/7663 - 26s - loss: 0.1970 - val_loss: 3.6396
Epoch 164/200
7663/7663 - 25s - loss: 0.2161 - val_loss: 3.4474
Epoch 165/200
7663/7663 - 28s - loss: 0.2356 - val_loss: 3.2350
Epoch 166/200
7663/7663 - 34s - loss: 0.2083 - val_loss: 3.8221
Epoch 167/200
7663/7663 - 28s - loss: 0.2118 - val_loss: 3.0664
Epoch 168/200
7663/7663 - 31s - loss: 0.2221 - val_loss: 3.4429
Epoch 169/200
7663/7663 - 30s - loss: 0.2074 - val_loss: 3.3265
Epoch 170/200
7663/7663 - 25s - loss: 0.2062 - val_loss: 3.8388
Epoch 171/200
7663/7663 - 23s - loss: 0.2270 - val_loss: 3.3013
Epoch 172/200
7663/7663 - 24s - loss: 0.1861 - val_loss: 3.2664
Epoch 173/200
7663/7663 - 23s - loss: 0.2241 - val_loss: 3.6840
Epoch 174/200
7663/7663 - 23s - loss: 0.2072 - val_loss: 3.1935
Epoch 175/200
7663/7663 - 23s - loss: 0.2121 - val_loss: 3.1314
Epoch 176/200
7663/7663 - 22s - loss: 0.2108 - val_loss: 3.3659
Epoch 177/200
7663/7663 - 21s - loss: 0.2075 - val_loss: 3.1343
Epoch 178/200
7663/7663 - 21s - loss: 0.2183 - val_loss: 3.1087
Epoch 179/200
7663/7663 - 22s - loss: 0.2143 - val_loss: 3.9148
Epoch 180/200
7663/7663 - 21s - loss: 0.2014 - val_loss: 3.0298
Epoch 181/200
7663/7663 - 21s - loss: 0.2150 - val_loss: 3.5657
Epoch 182/200
7663/7663 - 21s - loss: 0.2019 - val_loss: 3.6403
Epoch 183/200
7663/7663 - 22s - loss: 0.2042 - val_loss: 3.6618
Epoch 184/200
7663/7663 - 26s - loss: 0.2044 - val_loss: 3.6667
Epoch 185/200
7663/7663 - 23s - loss: 0.2032 - val_loss: 3.1309
Epoch 186/200
7663/7663 - 23s - loss: 0.1997 - val_loss: 3.3026
```

```
Epoch 187/200
7663/7663 - 25s - loss: 0.2061 - val_loss: 3.4192
Epoch 188/200
7663/7663 - 21s - loss: 0.2180 - val_loss: 3.4246
Epoch 189/200
7663/7663 - 21s - loss: 0.1985 - val_loss: 3.5157
Epoch 190/200
7663/7663 - 21s - loss: 0.2062 - val_loss: 3.2458
Epoch 191/200
7663/7663 - 21s - loss: 0.2082 - val_loss: 3.3201
Epoch 192/200
7663/7663 - 21s - loss: 0.2028 - val_loss: 3.7915
Epoch 193/200
7663/7663 - 21s - loss: 0.2253 - val_loss: 3.5342
Epoch 194/200
7663/7663 - 21s - loss: 0.1961 - val_loss: 3.6840
Epoch 195/200
7663/7663 - 21s - loss: 0.2079 - val_loss: 3.8383
Epoch 196/200
7663/7663 - 22s - loss: 0.1948 - val_loss: 3.8454
Epoch 197/200
7663/7663 - 23s - loss: 0.2153 - val_loss: 3.4546
Epoch 198/200
7663/7663 - 25s - loss: 0.2037 - val_loss: 3.6156
Epoch 199/200
7663/7663 - 22s - loss: 0.2097 - val_loss: 3.4659
Epoch 200/200
7663/7663 - 25s - loss: 0.1900 - val_loss: 3.7400
```

[217]:

```
# Com a função de perda MSE 500 cópia
pyplot.plot(history.history['loss'], label='treinamento')
pyplot.plot(history.history['val_loss'], label='teste')
pyplot.legend()
pyplot.show()
```



[218]:
```
# definir um modelo de codificador (sem o decodificador)
encoder = Model(inputs=visible, outputs=bottleneck)
```

[219]:
```
# salvo o encoder para usar depois
encoder.save('encoder_projeto_200.h5')
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet
to be built. `model.compile_metrics` will be empty until you train or evaluate
the model.
```

7.1 Treinando um modelo Random Forest com a rede neural

[220]:
```
# Carrega o modelo
from tensorflow.keras.models import load_model
encoder = load_model('encoder_projeto_200.h5')
```

```
WARNING:tensorflow:No training configuration found in the save file, so the
model was *not* compiled. Compile it manually.
```

[221]:
```
# Treinando no encoder
X_train_encode = encoder.predict(X)
# encode the test data
X_test_encode = encoder.predict(X_teste)
```

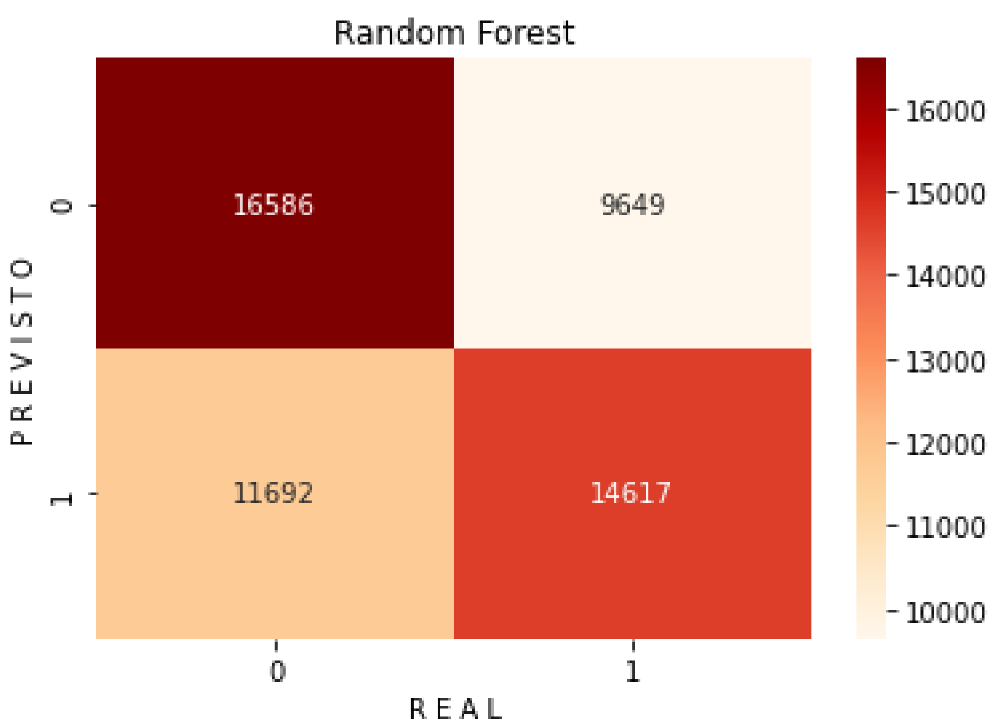
[222]:
```
#Define o modelo copia MSE
```

```
floresta = RandomForestClassifier(n_estimators = 10, criterion =␣
 ↪'entropy',random_state=123)
#Ajuste do modelo do conjunto de treinamento
floresta.fit(X_train_encode,y)
#Faz a predição no conjunto de teste
pred_rf = floresta.predict(X_test_encode)
#Calcula accuracy
acc = accuracy_score(y_teste,pred_rf)
print(acc)
```

0.5938451583434835

7.2 Calculando as métricas

[223]:
```
#Matriz de confusão
sns.heatmap(confusion_matrix(y_teste, pred_rf), 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()
```



[225]:
```
#Acurácia, Sensibilidade positiva (VP/(VP+FN), Especificidade, Precisão, Recall,␣
↪F1-Score
acuracia_rf_rede = accuracy_score(y_teste,pred_rf)
especificidade_rf_rede = specificity_score(y_teste,pred_rf)
precisao_rf_rede = precision_score(y_teste,pred_rf)
recall_rf_rede = recall_score(y_teste,pred_rf)
f1Score_rf_rede = f1_score(y_teste,pred_rf)
curva_roc_escore_rf_rede = roc_auc_score(y_teste,pred_rf)
kappa_rf_rede = cohen_kappa_score(y_teste,pred_rf)
print(f'Acurácia:{round(acuracia_rf_rede,2)}')
print(f'Especificidade:{round(especificidade_rf_rede,2)}')
print(f'Precisão:{round(acuracia_rf_rede,2)}')
print(f'Recall ou Sensibilidade:{round(recall_rf_rede,2)}')
print(f'F1-Score:{round(f1Score_rf_rede,2)}')
print(f'Kappa:{round(kappa_rf_rede,2)}')
print(f'Curva ROC:{round(curva_roc_escore_rf_rede,2)}')
```

```
Acurácia:0.59
Especificidade:0.63
Precisão:0.59
Recall ou Sensibilidade:0.56
F1-Score:0.58
Kappa:0.19
Curva ROC:0.59
```

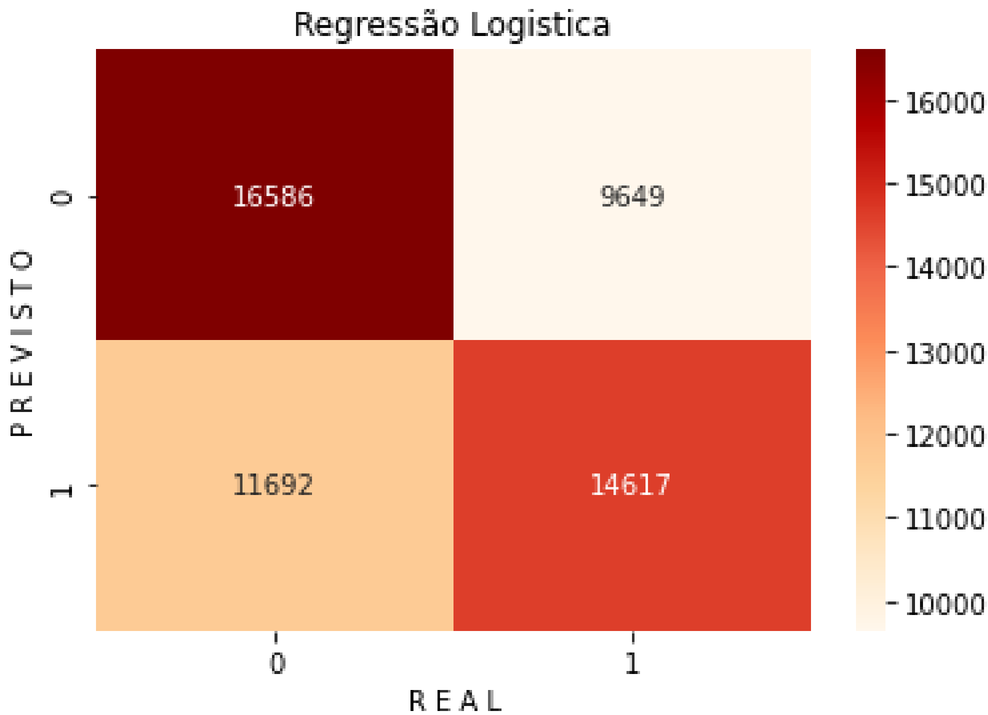
7.3 Treinando um modelo de Regressão Logística com a rede neural.

[226]:
```
#Define o modelo
model = LogisticRegression(max_iter=2000)
#Ajuste do modelo do conjunto de treinamento
model.fit(X_train_encode,y)
#Faz a predição no conjunto de teste
pred_rl = model.predict(X_test_encode)
#Calcula accuracy
acc = accuracy_score(y_teste,pred_rl)
print(acc)
```

```
0.5233137941534713
```

7.4 Calculando as métricas

[227]:
```
#Matriz de confusão
sns.heatmap(confusion_matrix(y_teste, pred_rf), cmap='OrRd', annot=True, fmt='2.
↪0f')
plt.title('Regressão Logistica')
plt.ylabel('P R E V I S T O')
plt.xlabel('R E A L')
plt.show()
```



[228]:
```
#Acurácia, Sensibilidade positiva (VP/(VP+FN), Especificidade, Precisão, Recall,␣
↪F1-Score
acuracia_rl_rede = accuracy_score(y_teste,pred_rl)
especificidade_rl_rede = specificity_score(y_teste,pred_rl)
precisao_rl_rede = precision_score(y_teste,pred_rl)
recall_rl_rede = recall_score(y_teste,pred_rl)
f1Score_rl_rede = f1_score(y_teste,pred_rl)
curva_roc_escore_rl_rede = roc_auc_score(y_teste,pred_rl)
kappa_rl_rede = cohen_kappa_score(y_teste,pred_rl)
print(f'Acurácia:{round(acuracia_rl_rede,2)}')
print(f'Especificidade:{round(especificidade_rl_rede,2)}')
print(f'Precisão:{round(precisao_rl_rede,2)}')
print(f'Recall ou Sensibilidade:{round(recall_rl_rede,2)}')
print(f'F1-Score:{round(f1Score_rl_rede,2)}')
print(f'Kappa:{round(kappa_rl_rede,2)}')
```

```
print(f'Curva ROC:{round(curva_roc_escore_rl_rede,2)}')
```

```
Acurácia:0.52
Especificidade:0.73
Precisão:0.54
Recall ou Sensibilidade:0.32
F1-Score:0.4
Kappa:0.05
Curva ROC:0.52
```

7.5 Curva ROC: Wrapper X Filtro X Autoencoder

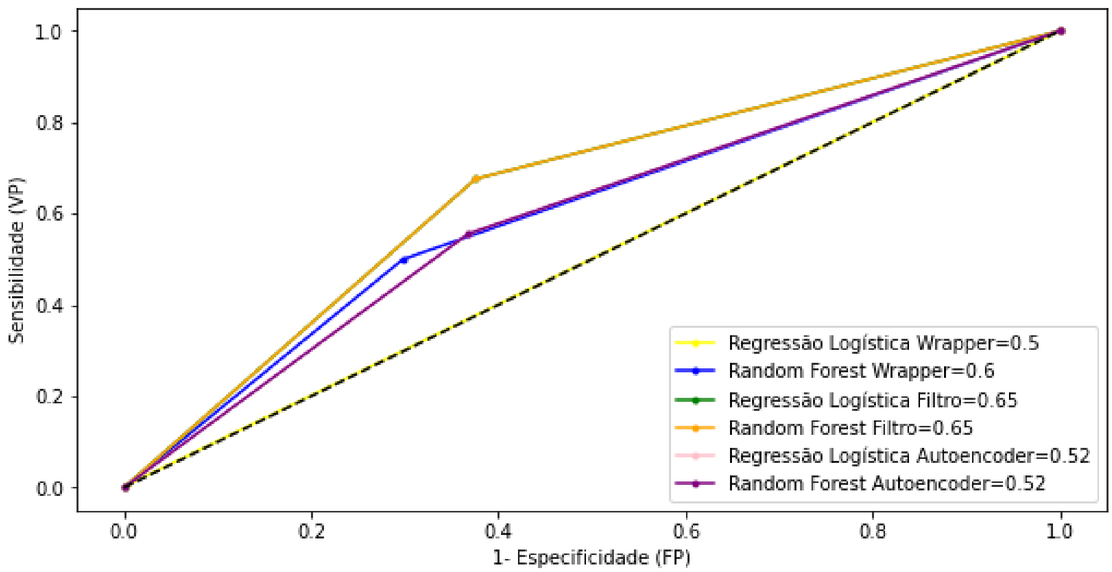
[231]:
```
#Wrapper
rfp_rl, rvp_rl,lim1 = roc_curve(y_teste,rl_pred)
pyplot.plot(rfp_rl, rvp_rl, marker='.', label='Regressão Logística␣
 ↪Wrapper='+str(round(curva_roc_escore_rl,2)),color='yellow')
rfp_rf, rvp_rf,lim2 = roc_curve(y_teste,rf_pred)
pyplot.plot(rfp_rf, rvp_rf, marker='.', label='Random Forest␣
 ↪Wrapper='+str(round(curva_roc_escore_rf,2)),color="blue")

#Filtro
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␣
 ↪Filtro='+str(round(curva_roc_escore_rl_f,2)),color='green')
pyplot.plot(rfp_rf_f, rvp_rf_f, marker='.', label='Random Forest␣
 ↪Filtro='+str(round(curva_roc_escore_rf_f,2)),color='orange')



#Autoencoder

rfp_rl_rede, rvp_rl_rede,lim7 = roc_curve(y_teste,pred_rl)
rfp_rl_rede, rvp_rl_rede,lim8 = roc_curve(y_teste,pred_rf)
plt.plot(rfp_rl_rede, rvp_rl_rede, marker='.', label='Regressão Logística␣
 ↪Autoencoder='+str(round(curva_roc_escore_rl_rede,2)),color='pink')
plt.plot(rfp_rl_rede, rvp_rl_rede, marker='.', label='Random Forest␣
 ↪Autoencoder='+str(round(curva_roc_escore_rl_rede,2)),color='purple')
plt.plot([0, 1], [0, 1], color='black', linestyle='--')
# alterando o nome dos eixos
plt.xlabel('1- Especificidade (FP)')
plt.ylabel('Sensibilidade (VP)')
plt.legend()
# Mostrando o gráfico
plt.rcParams["figure.figsize"] = (15, 5)
plt.show()
```



7.6 Comparando as métricas

[232]:
```
Image(filename='tab_metricas.png')
```

[232]:

	Métodos					
Métrica	**RF Wrapper**	**RL Wrapper**	**RF Filter**	**RL Filter**	**RF Autoenc.**	**RL Autoenc.**
Acurácia	60	50	65	65	60	52
Especificidade	70	100	62	62	63	73
Precisão	63	61	64	64	59	54
Recall ou Sensibilidade	50	0.01	68	68	56	32
F1-Score	56	0.01	66	66	58	40
Kappa	20	0	30	30	19	5

8 Referências

8.0.1 [1] MACHINELEARNINGMASTERY. Disponível em: https://machinelearningmastery.com/autoencode for-classification/.Acesso em 21/11/2021.

8.0.2 [2] Wang, J., & Wang, L. (2020). Prediction and prioritization of autism-associated long non-coding RNAs using gene expression and sequence features. BMC Bioinformatics, 21(1), 1–15. https://doi.org/10.1186/s12859-020-03843-5.

8.0.3 [3] MONOLITONIMBUS. Disponível em: https://www.monolitonimbus.com.br/modelo-sequencial-do-keras/. Acesso em: 21/11/2021.

8.0.4 [4] KERAS. Disponível em: https://keras.io/api/losses/. Acesso em 21/11/2021.

8.0.5 [5] KERAS-Optimizer. Disponível em: https://keras.io/guides/training_with_built_in_methods/. Acesso em 22/11/2021.