Grupo:

- Arthur Estevão
- Guilherme Giacomin
- Sílvia Fonseca
- Uriel Braga

Repositório: https://github.com/Arthuresm/Trabalho-Final-IA

Proposta de trabalho

Este trabalho tem como objetivo, realizar um estudo sobre o mercado de filmes listados no IMDb.

O estudo consiste em possibilitar criar um modelo baseado nos algoritmos ***Decision Tree Classifier*** e ***Random Forest***.

Ambos os algoritmos possuem um grau de assertividade normalmente alto para bases consideravelmente pequenas, o que não é o caso, visto que a quantidade de filmes e a diversificação entre eles é extremamente grande. Por conta disso a abordagem irá seguir avaliando quais características mais contribuem na geração de um bom modelo, na tentativa de criar um novo filme que pode ter uma nota satisfatória segundo a crítica.

```
#@markdown Temos 4 bases disponíveis (hospedadas no github) para realizar o estudo

data_urls = {
    "movies": "https://github.com/Arthuresm/Trabalho-Final-IA/raw/main/Data/IMDb%2
    "names": "https://github.com/Arthuresm/Trabalho-Final-IA/raw/main/Data/IMDb%20
    "ratings": "https://github.com/Arthuresm/Trabalho-Final-IA/raw/main/Data/IMDb%
    "titles": "https://github.com/Arthuresm/Trabalho-Final-IA/raw/main/Data/IMDb%2
}


import pandas as pd
import numpy as np


def get_data_frame_from_zip(url):
```

Temos 4 bases disponíveis (hospedadas no github) para realizar o estudo onde cada uma tem sua especificidade.

```
  return pd.read_csv(url, compression="zip")


dfs = {
    "movies": get_data_frame_from_zip(data_urls["movies"]),
    "names": get_data_frame_from_zip(data_urls["names"]),
    "ratings": get_data_frame_from_zip(data_urls["ratings"]),
    "titles": get_data_frame_from_zip(data_urls["titles"])
}
```

```
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2822: DtypeWarning:

Columns (3) have mixed types.Specify dtype option on import or set low_memory=False.
```

```
#@markdown Aqui podemos começar a visualizar quais as características de cada base
dfs["movies"].head()
```

Aqui podemos começar a visualizar quais as características de cada base.

	imdb_title_id	title	original_title	year	date_published	genre	duration	country	language	director	
0	tt0000009	Miss Jerry	Miss Jerry	1894	1894-10-09	Romance	45	USA	None	Alexander Black	A
1	tt0000574	The Story of the Kelly Gang	The Story of the Kelly Gang	1906	1906-12-26	Biography, Crime, Drama	70	Australia	None	Charles Tait	

```
dfs["names"].head()
```

	imdb_name_id	name	birth_name	height	bio	birth_details	date_of_birth	place_of_birth	death_details
0	nm0000001	Fred Astaire	Frederic Austerlitz Jr.	177.0	Fred Astaire was born in Omaha, Nebraska, to J...	May 10, 1899 in Omaha, Nebraska, USA	1899-05-10	Omaha, Nebraska, USA	June 22, 1987 in Los Angeles, California, USA ...
1	nm0000002	Lauren Bacall	Betty Joan Perske	174.0	Lauren Bacall was born Betty Joan Perske on Se...	September 16, 1924 in The Bronx, New York City...	1924-09-16	The Bronx, New York City, New York, USA	August 12, 2014 in New York City, New York, US...

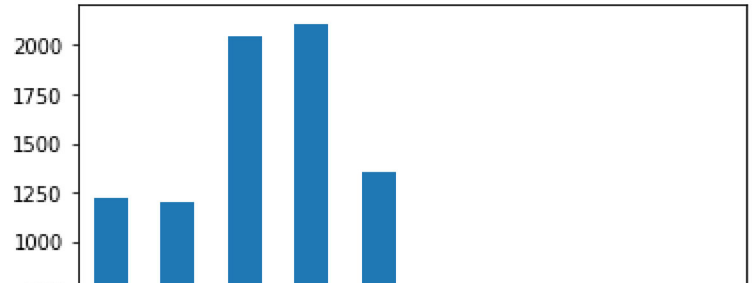
```
dfs["ratings"].head()
```

	imdb_title_id	weighted_average_vote	total_votes	mean_vote	median_vote	votes_10	votes_9	votes_8	votes_7	vote
0	tt0000009	5.9	154	5.9	6.0	12	4	10	43	
1	tt0000574	6.1	589	6.3	6.0	57	18	58	137	
2	tt0001892	5.8	188	6.0	6.0	6	6	17	44	
3	tt0002101	5.2	446	5.3	5.0	15	8	16	62	
4	tt0002130	7.0	2237	6.9	7.0	210	225	436	641	

Ernst

```
dfs["ratings"].mean(axis=0)[4:14].plot(kind="bar")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff7c59bcc50>
```




```
dfs["titles"].head()
```

	imdb_title_id	ordering	imdb_name_id	category	job	characters
0	tt0000009	1	nm0063086	actress	NaN	["Miss Geraldine Holbrook (Miss Jerry)"]
1	tt0000009	2	nm0183823	actor	NaN	["Mr. Hamilton"]
2	tt0000009	3	nm1309758	actor	NaN	["Chauncey Depew - the Director of the New Yor...
3	tt0000009	4	nm0085156	director	NaN	NaN
4	tt0000574	1	nm0846887	actress	NaN	["Kate Kelly"]

Definindo a base que vamos trabalhar

Estudando as bases, podemos perceber que a primeira contém grandes informações sobre os filmes, como ano, gênero, país, língua, etc. Além disso, ela condensa a maior parte dos dados contidos nas outras 3 bases.

Dese modo, essa será utilizada.

```
#@markdown Definindo as características que serão trabalhadas
retricted_columns = dfs["movies"][["year", "genre", "country", "language", "direct
```

Definindo as características que serão trabalhadas

```
#@markdown Buscando por dados inválidos na coluna year
count = 0

invalid_values = []
```

Buscando por dados inválidos na coluna year

```
for date in retricted_columns["year"]:
  if type(date) != int and len(date) != 4:
    invalid_values.append(count)
  count += 1
print(count)
```

```
85855
```

```
#@markdown O único dado inválido está na linha **83917**
invalid_values
```

O único dado inválido está na linha **83917**

```
[83917]
```

```
retricted_columns.loc[83917]
```

```
year                                                    TV Movie 2019
genre                                          Biography, Comedy, Crime
country                                                           USA
language                                                      English
director                                                  Cory Finley
writer                                       Mike Makowsky, Robert Kolker
production_company                                          HBO Films
actors                 Hugh Jackman, Ray Romano, Welker White, Alliso...
avg_vote                                                          7.1
Name: 83917, dtype: object
```

```
#@markdown Optamos por removê-lo
retricted_columns.drop(83917, axis=0, inplace=True)
```

Optamos por removê-lo

```
/usr/local/lib/python3.6/dist-packages/pandas/core/frame.py:4174: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-
```

```
#@markdown Transformando o tipo da coluna para dados numéricos
```

```
#@markdown Transformando o tipo da coluna para dados numéricos
retricted_columns["year"] = pd.to_numeric(retricted_columns["year"])
```

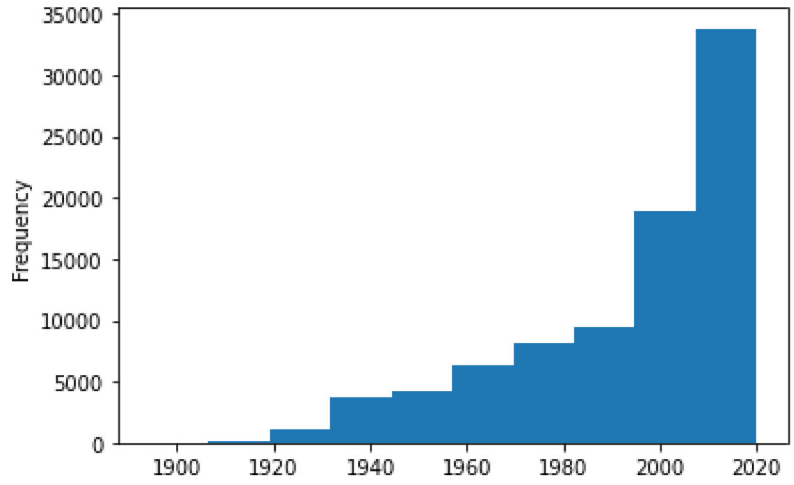
Transformando o tipo da coluna para dados numéricos

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-
```

```
#@markdown Observando os filmes produzidos ao longo dos anos
retricted_columns["year"].plot(kind="hist")
```

Observando os filmes produzidos ao longo dos anos

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff7c58b8c88>
```



Aqui realizamos uma decisão importante, onde os filmes que serão trabalhados foram lançados a partir de 1990. Isso foi feito pois filmes tem uma forte tendência temporal e uma quantidade significativa da base está descrita a partir desse ano.

```
retricted_columns = retricted_columns.sort_values(by=["year"], ascending=False)
```

```
retricted_columns = retricted_columns[retricted_columns["year"] >= 2010]
```

```
retricted_columns.reset_index(inplace=True)
```

```
retricted_columns.columns
retricted_columns.drop('index', axis=1, inplace=True)
```

```
#@markdown Se houver qualquer dado nulo na linha vamos removê-la
retricted_columns = retricted_columns.dropna(subset=["genre", "country", "language
```

Se houver qualquer dado nulo na linha vamos removê-la

```
#@markdown Número de filmes dentro de cada pontuação
retricted_columns["avg_vote"].value_counts()
```

Número de filmes dentro de cada pontuação

```
6.4    1011
6.2     987
6.3     976
6.1     930
6.5     920
       ...
9.2       3
9.8       2
9.7       2
9.4       1
9.5       1
Name: avg_vote, Length: 88, dtype: int64
```

```
#@markdown Média de pontuação por ano
retricted_columns.set_index("year").groupby(level="year").mean()
```

Média de pontuação por ano

year	avg_vote
2010	5.641568
2011	5.640528
2012	5.642936
2013	5.633959
2014	5.676943
2015	5.648309

```
#@markdown Descrevendo a base atual. Um detalhe importante está no número de filme
retricted_columns.describe()
```

Descrevendo a base atual. Um detalhe importante está no número de filmes (51886)

	year	avg_vote
count	26487.000000	26487.000000
mean	2014.867369	5.681617
std	2.880509	1.300296
min	2010.000000	1.000000
25%	2012.000000	4.900000
50%	2015.000000	5.900000
75%	2017.000000	6.600000
max	2020.000000	9.800000

Definindo as características trabalhadas

Os dados da base devem ser trabalhados para criarmos um modelo mais assertivo. Começaremos pela característica alvo, onde serão divididos intervalos para as notas serem classificadas.

```
# TERRIBLE = 1.0
# BAD = 2.0
# REGULAR = 3.0
# GREAT = 4.0
# EXCELLENT = 5.0

# TERRIBLE = "TERRIBLE"
BAD = "BAD"
REGULAR = "REGULAR"
GREAT = "GREAT"
# EXCELLENT = "EXCELLENT"


# retricted_columns['avg_vote_class'] = TERRIBLE

retricted_columns['avg_vote_class'] = BAD

# mask = (retricted_columns["avg_vote"] > 2) & (retricted_columns["avg_vote"] <= 4)
# retricted_columns.loc[mask, "avg_vote_class"] = BAD

mask = ((retricted_columns["avg_vote"] >= 5) & (retricted_columns["avg_vote"] <= 6))
retricted_columns.loc[mask, "avg_vote_class"] = REGULAR

mask = ((retricted_columns["avg_vote"] > 6) & (retricted_columns["avg_vote"] <= 10))
retricted_columns.loc[mask, "avg_vote_class"] = GREAT

# mask = (retricted_columns["avg_vote"] > 8)
# retricted_columns.loc[mask, "avg_vote_class"] = EXCELLENT
```

Diversas características apresentam um conjunto de possibilidades na base e com isso, vamos utilizar a técnica de One Hot Encoding para determinar em qual característica o filme se enquadra.

```
def sepate_data_by_comma(data):
  data_list = set()
  for e in data:
    i_begin = 0
```

```
      i_end = e.find(",", i_begin)
      while i_end != -1:
        data_list.add(e[i_begin:i_end:])
        i_begin = i_end + 2;
        i_end = e.find(",", i_begin)
      data_list.add(e[i_begin::])
    return data_list


def initialize_dict_with_valid_keys(valid_keys):
  result = {}
  for new_key in valid_keys:
    result[new_key] = []
  return result


def get_valid_keys(data, column_name):
  return sepate_data_by_comma(data[column_name])


def one_hot_encoding(df, column_name):
  valid_keys = get_valid_keys(df, column_name)

  result = initialize_dict_with_valid_keys(valid_keys)

  for index, row in df.iterrows():
    for key in valid_keys:
      result[key].append(1 if key in row[column_name] else 0)

  return result


def add_columns(df, new_df_columns):
  for new_key in new_df_columns.keys():
    df[new_key] = new_df_columns[new_key]
  return df


def one_hot_column(df, column_name):
  values_column = get_valid_keys(df, column_name)
```

```
  print("Hot start Column: ", column_name)
  data = one_hot_encoding(df, column_name)
  df_with_values_encoded = pd.DataFrame(data, columns=values_column, index=df.index)
  df_result = df
  df_result = add_columns(df_result, df_with_values_encoded)
  return df_result


def one_hot_list(df, column_list):
  for column_name in column_list:
    df = one_hot_column(df, column_name)
  return df


retricted_columns.head()
```

	year	genre	country	language	director	writer	production_company	actors	avg_vote	avg_vote_class
1	2020	Horror, Mystery	USA, Canada	English	Nicolas Pesce	Nicolas Pesce, Nicolas Pesce	Screen Gems	Tara Westwood, Junko Bailey, David Lawrence Br...	4.2	BAD
2	2020	Action, Crime, Drama	USA	English	Prince Bagdasarian	Prince Bagdasarian	Imaginating Pictures	Scout Taylor-Compton, Michael Urie, Najarra	4.2	BAD

Aqui, determinamos quais características serão avaliadas para o modelo de One Hot.

```
column_list = ["genre", "language"]

#"genre", "country", "language", "director", "writer", "production_company"

films_base = one_hot_list(retricted_columns, column_list)
```

```
Hot start Column:  genre
Hot start Column:  language
```

```
#@markdown Vamos visualizar as alterações realizadas.
films_base
```

Vamos visualizar as alterações realizadas.

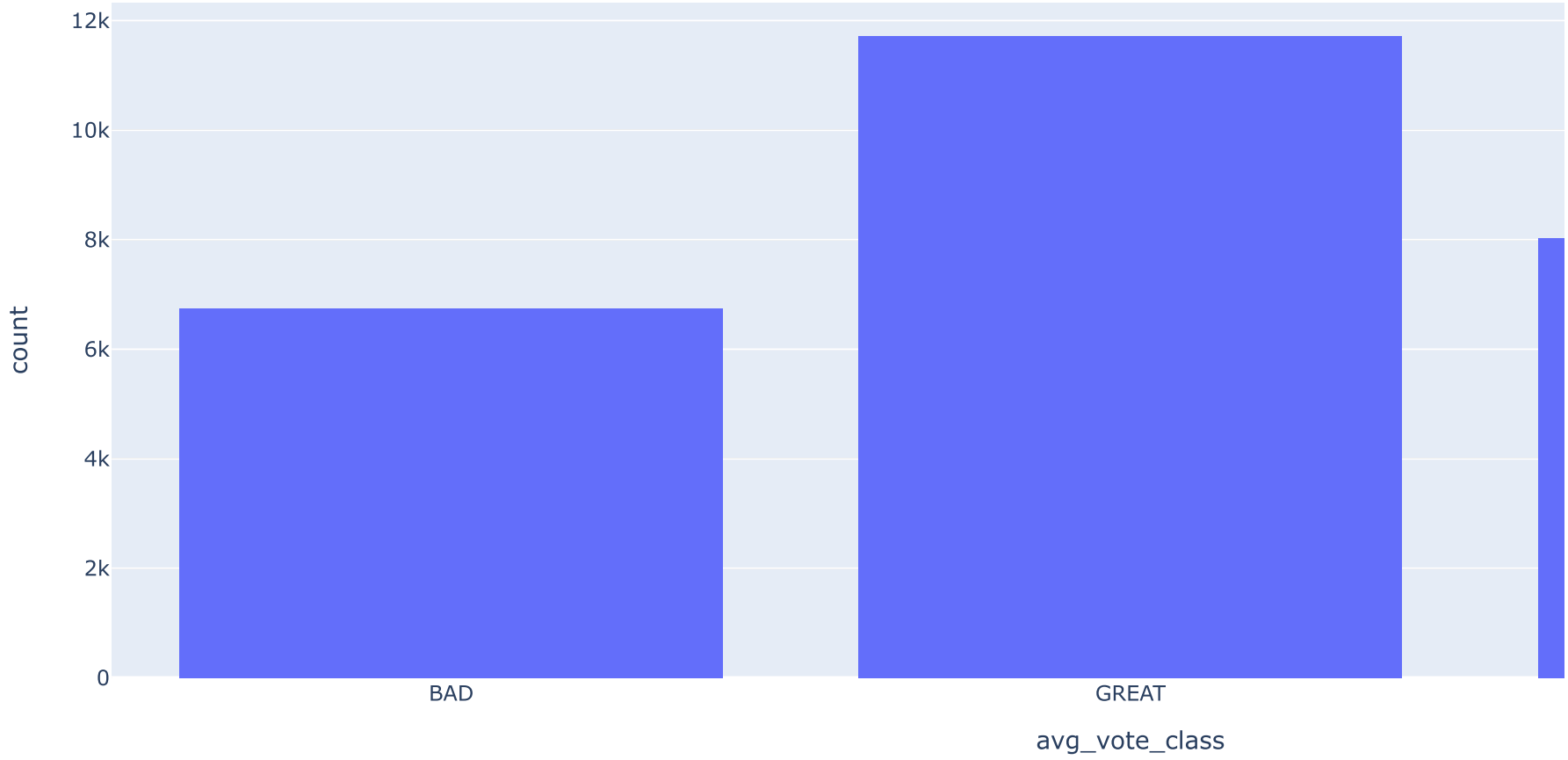
	year	genre	country	language	director	writer	production_company	actors	avg_vote	avg_vot
1	2020	Horror, Mystery	USA, Canada	English	Nicolas Pesce	Nicolas Pesce, Nicolas Pesce	Screen Gems	Tara Westwood, Junko Bailey, David Lawrence Br...	4.2	
2	2020	Action, Crime, Drama	USA	English	Prince Bagdasarian	Prince Bagdasarian	Imaginating Pictures	Scout Taylor-Compton, Michael Urie, Najarra To...	4.2	
3	2020	Drama	Italy	Italian	Francesco Amato	Francesco Amato, Massimo Gaudioso	Lucky Red	Vittoria Puccini, Benedetta Porcaroli, Edoardo...	6.7	
4	2020	Comedy	Italy	Italian	Giampaolo Morelli	Gianluca Ansanelli, Giampaolo Morelli	Italian International Film	Giampaolo Morelli, Serena Rossi, Diana Del Buf...	5.9	RI
5	2020	Action, Crime, Drama	USA	English	Steve Rahaman	Steve Rahaman	Rahaman Studios	Daniel O'Shea, Debra Toscano, Michael Robert A...	4.5	
...	...	...	...	...	...	...	...	...	...	...

```
#@markdown Aqui temos a apresentação dos rótulos e sua proporção na base. É possív
import plotly.express as px

fig = px.histogram(films_base, x="avg_vote_class")
```

Aqui temos a apresentação dos rótulos e sua proporção na base. É possível perceber que não temos nenhum rótulo se

```
fig.show()
```

destacando dos demais, representando um bom balancemento para a base trabalhada.




```
def get_columns_result():
  result = []
  for column_name in column_list:
    result += list(get_valid_keys(films_base, column_name))
  return result
```

```
columns = get_columns_result()

predictor = films_base.loc[:, columns].values
valid_class = films_base.iloc[:, 9].values


from sklearn.model_selection import train_test_split

predictor_training, predictor_test, class_training, class_test = train_test_split(predictor, valid_class, test_size=0.15, ra
```

Utilizando um modelo de árvore de decisão

O exemplo a seguir utiliza o algoritmo de árvore de decisão e possui como medida de qualidade da divisão do nó, a entropia.

```
from sklearn.tree import DecisionTreeClassifier

classifier = DecisionTreeClassifier(criterion= 'entropy',
                                    random_state=0)
classifier.fit(predictor_training, class_training)
previsions = classifier.predict(predictor_test)


from sklearn.metrics import confusion_matrix, accuracy_score
precision = accuracy_score(class_test, previsions)
matrix = confusion_matrix(class_test, previsions)
```

A precisão e matriz de confusão são apresentadas em sequência.

```
precision
```

```
0.546804227478611
```

```
matrix
```

```
array([[ 601,  223,  167],
       [ 193, 1240,  343],
       [ 338,  537,  332]])
```

Utilizando o algoritmo de Florestas aleatórias

Para utilizar o Random Forest, é necessário ter em mente o método de avaliação para o conjunto de testes.

Temos algumas opções a seguir, mas nem todas podem ser utilizadas, já que o problema em questão é multiclasse (temos filmes ruins, regulares e ótimos).

```
#@markdown Métricas e pontuações validas
import sklearn
sorted(sklearn.metrics.SCORERS.keys())
```

Métricas e pontuações validas

```
['accuracy',
 'adjusted_mutual_info_score',
 'adjusted_rand_score',
 'average_precision',
 'balanced_accuracy',
 'completeness_score',
 'explained_variance',
 'f1',
 'f1_macro',
 'f1_micro',
 'f1_samples',
 'f1_weighted',
 'fowlkes_mallows_score',
 'homogeneity_score',
 'jaccard',
 'jaccard_macro',
 'jaccard_micro',
 'jaccard_samples',
 'jaccard_weighted',
 'max_error',
 'mutual_info_score',
 'neg_brier_score',
 'neg_log_loss',
```

```
        'neg_mean_absolute_error',
        'neg_mean_gamma_deviance',
        'neg_mean_poisson_deviance',
        'neg_mean_squared_error',
        'neg_mean_squared_log_error',
        'neg_median_absolute_error',
        'neg_root_mean_squared_error',
        'normalized_mutual_info_score',
        'precision',
        'precision_macro',
        'precision_micro',
        'precision_samples',
        'precision_weighted',
        'r2',
        'recall',
        'recall_macro',
        'recall_micro',
        'recall_samples',
        'recall_weighted',
        'roc_auc',
        'roc_auc_ovo',
        'roc_auc_ovo_weighted',
        'roc_auc_ovr',
        'roc_auc_ovr_weighted',
        'v_measure_score']
```

Para nos auxiliar quanto aos hiperparâmetros que podem ser utilizados, vamos fazer uso do GridSearchCV.

Este é um grande módulo que possibilita gerar combinações para os hiperparâmetros desejados.

```
#@markdown Tipos de score disponíveis em [Sckit Learning Metrics](https://scikit-l
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

param_grid = [
              {'n_estimators': [1, 2, 3, 4, 5], "max_depth": [5, 10, 15, 25], 'cri
              {'bootstrap': [False], 'n_estimators': [1, 2, 3, 4, 5], 'max_feature
              {'n_estimators': [1, 10, 100, 200, 500, 1000], "max_depth": [5, 10],
]
```

Tipos de score disponíveis em Sckit Learning Metrics

```
random_forest = RandomForestClassifier()
```

Essa é a parte mais demorada, pois vamos executar todas as combinações possíveis para os hiperparâmtros definidos anteriormente.

O melhor modelo estará presente no fim da execução.

```
scoring_grid = ['f1_macro', 'f1_micro', 'precision_micro', 'recall_macro','recall_micro']
grid_search = None
best_grid = 0
for current_scoring in scoring_grid:
  print(current_scoring)
  current_grid_search = GridSearchCV(random_forest, param_grid, cv=5,   #Utilizando validacao cruzada de 5 partes
                           scoring=current_scoring)
  current_grid_search.fit(predictor, valid_class)
  if current_grid_search.best_score_ > best_grid:
    best_grid = current_grid_search.best_score_
    grid_search = current_grid_search
```

```
f1_macro
f1_micro
precision_micro
recall_macro
recall_micro
```

```
grid_search.cv_results_
```

```
        'random_state': 42},
       {'bootstrap': False,
        'criterion': 'entropy',
        'max_depth': 5,
        'max_features': 2,
        'n_estimators': 1,
        'random_state': 42},
       {'bootstrap': False,
        'criterion': 'entropy',
        'max_depth': 5,
        'max_features': 2,
```

```
 'n_estimators': 2,
 'random_state': 42},
{'bootstrap': False,
 'criterion': 'entropy',
 'max_depth': 5,
 'max_features': 2,
 'n_estimators': 3,
 'random_state': 42},
{'bootstrap': False,
 'criterion': 'entropy',
 'max_depth': 5,
 'max_features': 2,
 'n_estimators': 4,
 'random_state': 42},
{'bootstrap': False,
 'criterion': 'entropy',
 'max_depth': 5,
 'max_features': 2,
 'n_estimators': 5,
 'random_state': 42},
{'bootstrap': False,
 'criterion': 'entropy',
 'max_depth': 5,
 'max_features': 3,
 'n_estimators': 1,
 'random_state': 42},
{'bootstrap': False,
 'criterion': 'entropy',
 'max_depth': 5,
 'max_features': 3,
 'n_estimators': 2,
 'random_state': 42},
{'bootstrap': False,
 'criterion': 'entropy',
 'max_depth': 5,
 'max_features': 3,
 'n_estimators': 3,
 'random_state': 42},
{'bootstrap': False,
 'criterion': 'entropy',
 'max_depth': 5,
 'max_features': 3,
 'n_estimators': 4,
 'random_state': 42}
```

```
 random_state : 42},
{'bootstrap': False,
 'criterion': 'entropy',
 'max_depth': 5,
 'max_features': 3,
 'n_estimators': 5,
```

Esses foram os melhores parâmetros encontrados para o problema (dentre os definidos).

- criterion: Mede critério de divisão do nó.
- max_depth: Profundidade máxima das árvores.
- n_estimators: Número de árvores de decisão na floresta.
- random_state: Controla a aleatoriedade na geração de amostras.

```
grid_search.best_params_
```

```
{'criterion': 'entropy',
 'max_depth': 25,
 'n_estimators': 5,
 'random_state': 42}
```

```
grid_search.estimator
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

```
best_random_forest = grid_search.best_estimator_
```

```
print(best_random_forest)
```

```
tree_example = best_random_forest.estimators_[0]
```

```
print(tree_example)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='entropy', max_depth=25, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=5,
                       n_jobs=None, oob_score=False, random_state=42, verbose=0,
                       warm_start=False)
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=25, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=1608637542, splitter='best')
```

Podemos perceber que o ganho não foi significativo pela demora no processo de validação e contrução do modelo.

Isso decorre da complexidade do problema trabalhado (as características utilizadas possivelmente não possuem uma forte correlação com a pontuação).

Outro fator que corrobora o resultado obtido é a forma com que o problema foi montado, talvez a utilização de todas as possibilidades de características para a língua e gênero do filme, não se adequa muito bem ao modelo utilizado e se faz necessário um algoritmo mais robusto.

```
print(grid_search.scorer_)
print(grid_search.best_score_)
```

```
make_scorer(f1_score, pos_label=None, average=micro)
0.5564992057656659
```

```
feature_importances = best_random_forest.feature_importances_
feature_importances
```

```
array([1.47987661e-01, 1.64202024e-02, 4.06180752e-03, 4.26170902e-03,
       1.51764544e-02, 1.04152385e-01, 1.79895784e-02, 2.24567583e-02,
```

```
3.46404463e-02, 2.45318702e-02, 1.89122773e-02, 0.00000000e+00,
2.41887887e-02, 3.90632786e-02, 3.34861697e-02, 9.37605707e-06,
1.97033269e-02, 4.28115763e-03, 1.20432653e-02, 9.01132122e-03,
9.43493850e-03, 1.93635018e-03, 9.81905700e-06, 8.69116820e-05,
3.36941789e-04, 0.00000000e+00, 5.05703619e-05, 2.09987423e-04,
7.89218625e-05, 2.78626185e-04, 8.81635101e-04, 0.00000000e+00,
9.52768991e-05, 8.09578865e-06, 1.82920834e-04, 1.11703258e-04,
1.84997240e-05, 0.00000000e+00, 3.42989675e-04, 4.81893506e-03,
0.00000000e+00, 4.24241568e-04, 0.00000000e+00, 0.00000000e+00,
4.74366663e-03, 2.77093852e-03, 0.00000000e+00, 1.80128614e-05,
6.52222739e-05, 2.02249103e-04, 6.89906013e-05, 5.42251270e-03,
8.56557320e-04, 4.67219882e-04, 3.30258923e-04, 3.20912682e-04,
0.00000000e+00, 4.94824932e-04, 1.38887120e-03, 0.00000000e+00,
5.28914411e-06, 3.75631494e-05, 0.00000000e+00, 2.75177182e-04,
1.30605431e-04, 1.94311777e-04, 4.74689177e-04, 2.64325640e-04,
0.00000000e+00, 6.93320735e-03, 0.00000000e+00, 2.07705175e-03,
5.60857177e-03, 2.14823932e-02, 0.00000000e+00, 0.00000000e+00,
1.46605809e-05, 4.11670848e-03, 2.47738175e-05, 6.83497760e-05,
0.00000000e+00, 6.10043259e-03, 9.67390462e-04, 0.00000000e+00,
1.00778430e-04, 1.92654557e-04, 1.02496465e-02, 5.18130523e-05,
1.56198422e-04, 9.35913796e-03, 8.54989225e-04, 2.23893517e-05,
2.09093659e-03, 5.39360547e-04, 4.27732347e-04, 0.00000000e+00,
3.10315738e-04, 2.32102778e-03, 0.00000000e+00, 2.59157753e-03,
1.15022391e-04, 0.00000000e+00, 1.58298809e-05, 7.50945755e-04,
5.56806398e-05, 1.78540843e-04, 1.56795663e-04, 6.82069673e-03,
1.09637378e-03, 5.37194853e-04, 7.84816318e-04, 3.70875431e-04,
0.00000000e+00, 2.44793527e-05, 0.00000000e+00, 2.19107771e-04,
7.20759022e-05, 0.00000000e+00, 4.02630919e-04, 2.04437805e-03,
6.37634461e-05, 6.51361760e-04, 5.87195257e-05, 2.71463618e-03,
3.07536199e-05, 0.00000000e+00, 1.45043737e-04, 4.11980429e-04,
5.69967993e-03, 7.43633899e-03, 0.00000000e+00, 0.00000000e+00,
6.17745717e-03, 8.99886308e-06, 0.00000000e+00, 8.93294067e-05,
9.77928877e-04, 7.56028173e-03, 1.23450506e-04, 1.39217560e-04,
1.74548027e-05, 2.11312055e-04, 1.51748899e-04, 1.78307790e-04,
0.00000000e+00, 2.27910537e-04, 1.82313807e-04, 2.43732966e-04,
8.09351007e-05, 0.00000000e+00, 0.00000000e+00, 4.35495146e-04,
1.14298272e-04, 0.00000000e+00, 1.18486114e-03, 5.54794641e-06,
1.77969155e-04, 4.82704626e-05, 4.41696180e-04, 4.35169505e-03,
4.07685797e-03, 8.37840987e-05, 3.09633683e-03, 0.00000000e+00,
2.77920424e-04, 3.34270170e-03, 0.00000000e+00, 3.46591883e-05,
0.00000000e+00, 9.47711404e-04, 1.01233272e-03, 2.58975813e-03,
1.23299744e-02, 9.61586920e-04, 2.70040036e-04, 8.67646782e-03,
4.50822623e-03, 1.06318829e-03, 2.30968532e-04, 0.00000000e+00,
```

```
0.00000000e+00, 3.01975060e-04, 1.63973298e-02, 1.32178644e-04,
6.84266693e-05, 3.36929461e-05, 2.16856439e-05, 4.58564002e-06,
2.34202957e-04, 0.00000000e+00, 2.21267508e-04, 1.62185192e-04,
2.42134138e-04, 2.46837450e-03, 4.79229737e-04, 0.00000000e+00,
0.00000000e+00, 0.00000000e+00, 2.74960730e-05, 2.73059005e-03,
5.49341063e-03, 3.52104629e-05, 0.00000000e+00, 9.78286583e-05,
1.65445646e-03, 2.78408420e-04, 0.00000000e+00, 1.31889354e-04,
1.11647153e-03, 7.14012293e-04, 6.27324230e-06, 5.90850909e-05,
0.00000000e+00, 1.11029782e-02, 0.00000000e+00, 2.80504190e-04,
4.30519282e-05, 2.15274533e-04, 9.05900168e-04, 6.08943343e-04,
6.58567561e-05, 1.60022030e-02, 1.22209371e-04, 1.06923935e-03,
1.21148443e-04, 0.00000000e+00, 1.74733833e-02, 2.95525436e-04,
0.00000000e+00, 6.87023016e-05, 1.05029260e-01, 0.00000000e+00,
```

Considerando a floresta como um todo, as características mais relevantes foram:

```
sorted(zip(feature_importances, columns), reverse=True)
```

```
[(0.14798766096978042, 'Drama'),
 (0.10502926017209316, 'English'),
 (0.10415238465891297, 'Horror'),
 (0.03906327857707234, 'Thriller'),
 (0.03464044630163017, 'Comedy'),
 (0.03348616973595417, 'Action'),
 (0.02453187022350976, 'Biography'),
 (0.024196499143797078, 'French'),
 (0.024188788659350945, 'Romance'),
 (0.022456758281791923, 'Crime'),
 (0.021482393219941536, 'Japanese'),
 (0.0197033268875947, 'Mystery'),
 (0.018912277304051518, 'Adventure'),
 (0.017989578424027493, 'Animation'),
 (0.017473383286798674, 'German'),
 (0.01642020237106777, 'Fantasy'),
 (0.016397329846310333, 'Spanish'),
 (0.01600220303230842, 'Russian'),
 (0.015176454376779414, 'Sci-Fi'),
 (0.012329974443377526, 'Italian'),
 (0.012292361503345148, 'Korean'),
 (0.012043265336096853, 'Family'),
 (0.011102978204168371, 'Hindi'),
```

```
(0.010249646476291844, 'Turkish'),
(0.00943493850137807, 'History'),
(0.009359137960354832, 'Mandarin'),
(0.00901132122241159, 'Music'),
(0.008676467821784956, 'Arabic'),
(0.007560281731526797, 'Cantonese'),
(0.007436338990316675, 'Polish'),
(0.006933207345090496, 'Dutch'),
(0.0068206967339396575, 'Tamil'),
(0.006177457165657985, 'Chinese'),
(0.006100432594046699, 'Portuguese'),
(0.00569967993267488, 'Telugu'),
(0.005608571771139526, 'Romanian'),
(0.005493410634671051, 'Indonesian'),
(0.0054225126983759395, 'Hebrew'),
(0.004818935063462882, 'Bengali'),
(0.004743666630836893, 'Swedish'),
(0.004695438027299016, 'Malay'),
(0.004508226228367996, 'Thai'),
(0.004351695046580895, 'Danish'),
(0.0042811576299178955, 'Sport'),
(0.004261709017568301, 'War'),
(0.004116708482080544, 'Persian'),
(0.0040768579655923905, 'Ukrainian'),
(0.004061807521749348, 'Musical'),
(0.0037950779802407636, 'Marathi'),
(0.003342701700549489, 'Urdu'),
(0.0030963368290423854, 'Norwegian'),
(0.0027709385233170994, 'Greek'),
(0.0027305900495824226, 'Serbian'),
(0.0027146361793378712, 'Hungarian'),
(0.002591577530818948, 'Punjabi'),
(0.0025897581266460007, 'Latin'),
(0.002468374497350746, 'Finnish'),
(0.002321027781626491, 'Vietnamese'),
(0.002162151821765879, 'Malayalam'),
```

A seguir iremos testar a aplicabilidade do modelo com o conjunto utilizado no exemplo de árvore de decisão. É importante ressaltar que esse teste não comprova a assetividade do modelo gerado. Exemplo de utilização do modelo

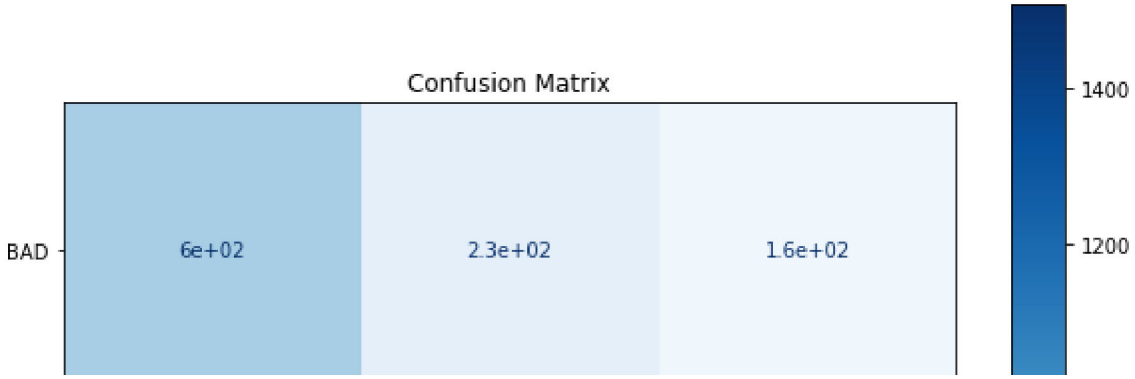
```
#@markdown Quantidade de elementos que serão testados por rótulo
from collections import Counter
```

Quantidade de elementos que serão testados por rótulo

```
from collections import Counter
Counter(class_test)
```

```
Counter({'BAD': 991, 'GREAT': 1776, 'REGULAR': 1207})
```

```
from sklearn.metrics import plot_confusion_matrix
import numpy as np
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(10, 10))
ax.set_title("Confusion Matrix")

plot_confusion_matrix(best_random_forest,
                      predictor_test,
                      class_test,
                      cmap=plt.cm.Blues,
                      ax=ax)
plt.show()
```




```
from sklearn.model_selection import cross_val_predict

y_scores = cross_val_predict(best_random_forest,
                             predictor_test,
                             class_test,
                             cv = 5)
```

Uma outra possibilidade seria reduzir o escopo do problema, gerando um modelo capaz de classificar um filme como excelente ou não. Com isso teríamos um problema binário e uma boa métrica seria a Curva ROC, que possibilita avaliar classificadores desse tipo. (O caso a seguir é só um exemplo de utilização, para torná-lo um caso completo deveria ser gerada uma nova base e um novo modelo)

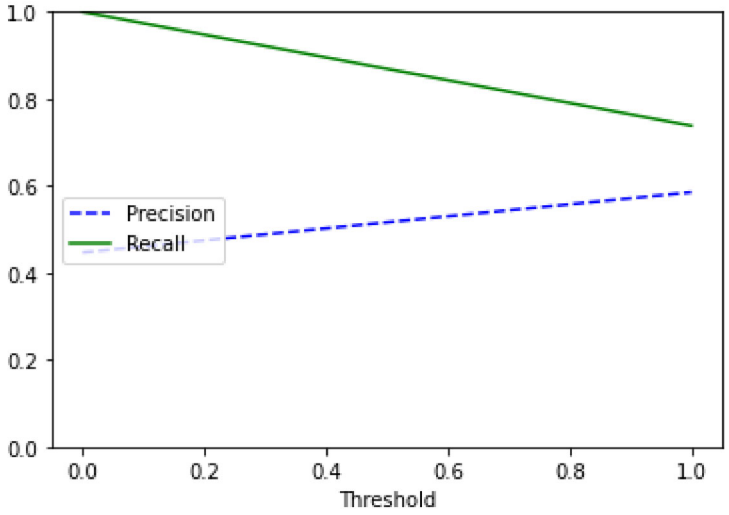
```
from sklearn.metrics import precision_score, recall_score, precision_recall_curve

precisions, recalls, thresholds = precision_recall_curve((class_test==GREAT), (y_scores==GREAT))
```

```
precisions
```

```
array([0.44690488, 0.58526786, 1.        ])
```

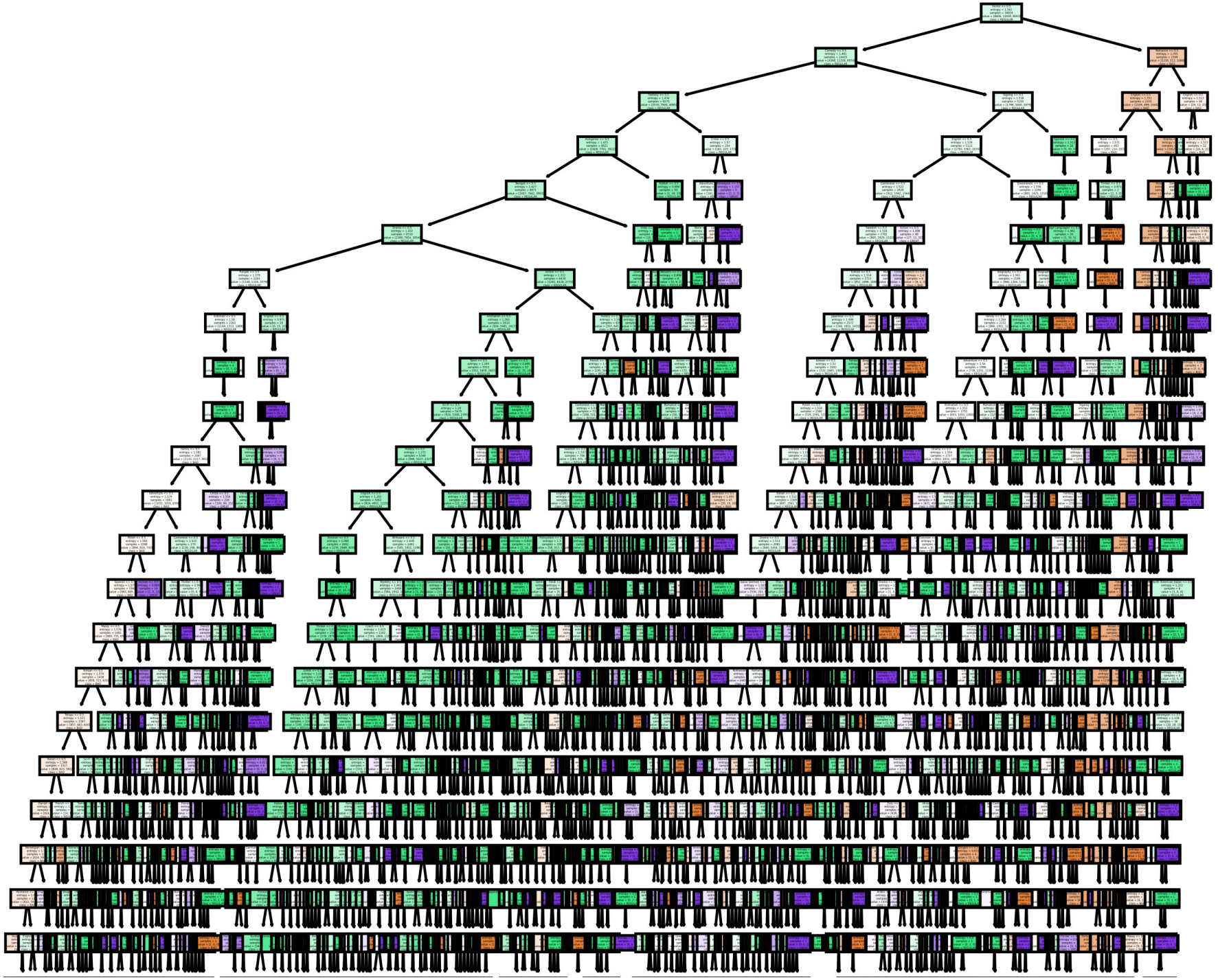
```
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
  plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
  plt.plot(thresholds, recalls[:-1], "g-", label="Recall")
  plt.xlabel("Threshold")
```

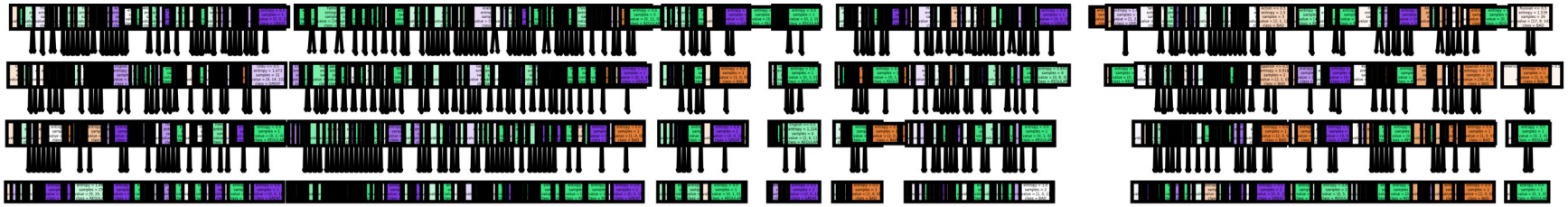
```
  plt.legend(loc="center left")
  plt.ylim([0, 1])


plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
plt.show()
```



Visualizando uma das árvores geradas na floresta.

```
from sklearn import tree

fn=columns

cn=[BAD, REGULAR, GREAT]

fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (8,8), dpi=1200)
tree.plot_tree(best_random_forest.estimators_[0],
               feature_names = fn,
               class_names=cn,
               filled = True);
fig.savefig('rf_individualtree.png')
```

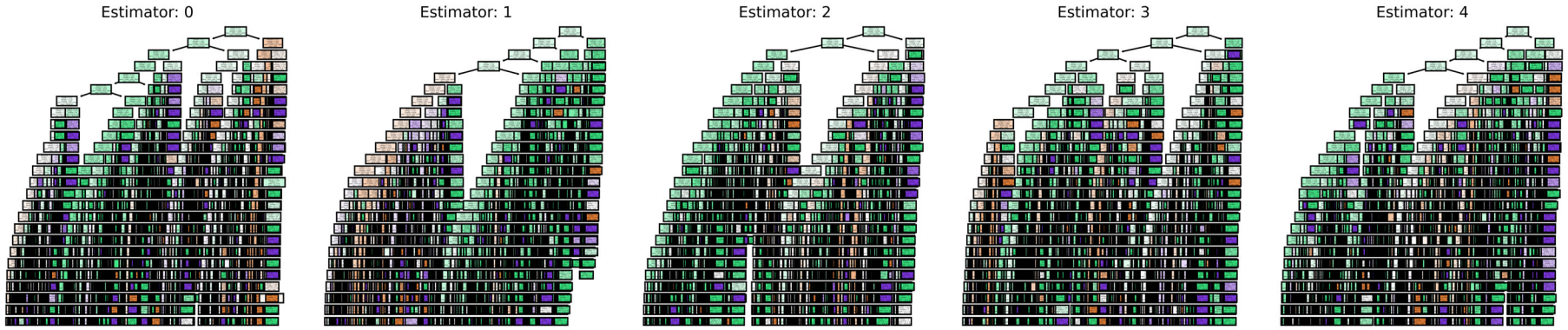




Percebendo algumas diferenças entre as árvores geradas.

```
total_trees = grid_search.best_params_["n_estimators"]

if total_trees > 5:
  total_trees = 5

fig, axes = plt.subplots(nrows = 1,ncols = total_trees,figsize = (20,4), dpi=900)
for index in range(0, total_trees):
    tree.plot_tree(best_random_forest.estimators_[index],
                   feature_names = fn,
```

```
                    class_names=cn,
                    filled = True,
                    ax = axes[index]);

    axes[index].set_title('Estimator: ' + str(index), fontsize = 11)
fig.savefig('rf_5trees.png')
```




```
# from google.colab import files
# files.download("rf_individualtree.png")
```